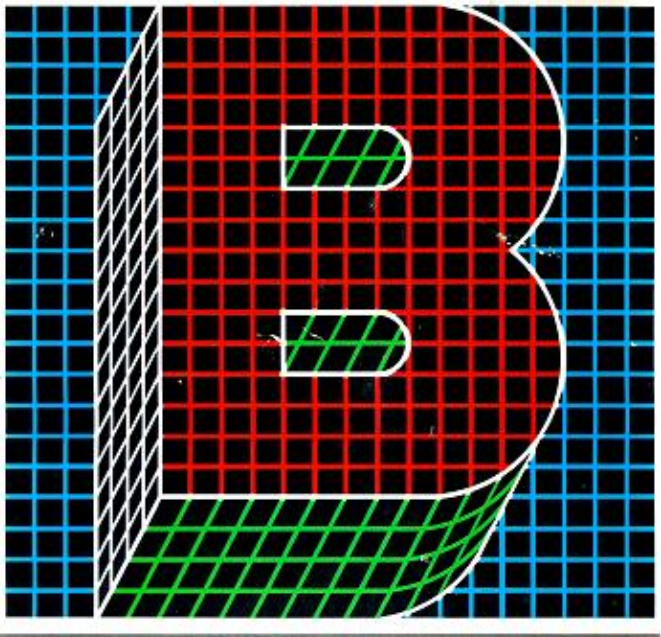
TO7



BASIC

Manuel de référence

RB.0082

TO TEK International

TO 7/70

MANUEL DE REFERENCE BASIC

TO TEK International

Ce volume porte la référence
ISBN 2-86848 - 000 - 4

Toute reproduction, même partielle, de cet ouvrage est interdite. Une copie ou reproduction par quelque procédé que ce soit, photographie, photocopie, microfilm, bande magnétique, disque ou autre, constitue une contrefaçon passible des peines prévues par la loi du 11 mars 1975 sur la protection des droits d'auteur.

Copyright © 1984. TO TEK International - Tour Gallieni 2.
36 Avenue Gallieni - 93175 BAGNOLET CEDEX FRANCE

SOMMAIRE

Pages

AVANT-PROPOS

I – GENERALITES 5

II – COMMANDES ET INSTRUCTIONS 25

III – FONCTIONS 109

IV – ENTREES/SORTIES ET FICHIERS 155

V – GRAPHISME 183

VI – SON 201

VII – EXEMPLES DE PROGRAMMES 205

ANNEXE A : REPRESENTATION DES VARIABLES EN MEMOIRE 211

ANNEXE B : FONCTIONS MATHEMATIQUES 213

ANNEXE C : ERREURS DETECTEES 215

ANNEXE D : MOTS RESERVES 217

ANNEXE E : JEU DE CODES ASCII 218

ANNEXE E bis : CODES ECRAN SPECIAUX 219

ANNEXE F : ORGANISATION DE LA MEMOIRE 220

ANNEXE G : LES 16 COULEURS DU T07-70 ET L'INCRUSTATION D'IMAGES VIDÉO 221

AVANT-PROPOS

Le **T07** dispose d'un langage **BASIC** à la fois très souple et très puissant, conçu pour exploiter au maximum toutes les possibilités du système

Outre les fonctions classiques du **BASIC**, des instructions complémentaires ont été développées pour donner accès très simplement aux multiples particularités du **T07** : éditeur plein écran, graphisme, couleur, musique

Cet ouvrage offre une description très détaillée du langage, illustrée de nombreux exemples. Conçu comme un manuel de référence, il suppose connues les bases du langage **BASIC**, exposées dans le manuel d'initiation au **BASIC T07**.

Le **BASIC** décrit ici, est réalisé sous forme d'une cartouche "**mémo 7**" enfichable dans l'unité de base du **T07**. C'est un **BASIC** niveau 5 **MICROSOFT** ® développé pour le **T07** (version 1.Ø)

L'extension disquette du **T07** contient par ailleurs des instructions **BASIC** complémentaires dont les lecteurs trouveront la description dans le livre "**Le BASIC D.O.S. du T07**"

I - GENERALITES

I-1 INTRODUCTION

Ce chapitre présente toutes les informations d'ordre général concernant le Basic disponible sur la cartouche "mémo 7" de l'ordinateur **TO7**.

Les instructions, les commandes ainsi que les fonctions seront traitées dans les chapitres suivants.

I-2 MISE EN SERVICE

Pour utiliser l'interpréteur Basic, opérer de la façon suivante :

- enficher la cartouche "mémo 7" Basic dans la trappe prévue à cet effet,
- verrouiller le volet de la trappe,
- mettre le téléviseur puis l'ordinateur sous tension,
- répondre **1** au menu affiché sur l'écran du téléviseur.

Les messages **"BASIC Version 1.0 (c) Microsoft 1982"** puis **"OK"** s'affichent : l'interpréteur Basic est actif.

Si le lecteur-enregistreur de programmes est sous tension, raccordé à l'ordinateur et la touche ▶ enfoncée, la réponse **2** au menu proposé entraîne le chargement en mémoire centrale du premier programme lu sur la cassette et son exécution immédiate sauf si un lecteur de disquettes est également connecté.

I-3 CLAVIER - CARACTERES DE CONTROLE

Le rôle des touches de fonction spécifiques au langage BASIC sera seul décrit ci-dessous le clavier étant décrit dans le mode d'emploi de l'unité centrale.

Ces touches jouent un rôle particulier et permettent l'introduction de certains caractères de contrôle.

ENTREE : Indique à l'ordinateur que la ligne en cours est terminée et provoque la prise en compte de celle-ci.
L'action sur cette touche positionne le curseur au début de la ligne suivante. Elle correspond au caractère **"Retour de chariot"**.

STOP : Arrête l'exécution du programme en cours. Pour relancer le programme appuyer sur une quelconque des touches du clavier à l'exception de **CNT** **•** et **STOP**.

CNT : Cette touche **"CONTROLE"** doit être utilisée simultanément avec une autre touche dont elle en modifie le sens.

CNT et **C** introduit le caractère **"CONTROLE-C"**. Il provoque l'interruption de l'exécution du programme en cours ou de la numérotation automatique si celle-ci est en vigueur. La ligne en cours d'introduction n'est pas prise en compte ; l'ordinateur passe en "mode commande". Une demande d'interruption par **CONTROLE-C** faite pendant l'exécution d'une instruction n'est prise en compte qu'à la fin de celle-ci. Dans le cas d'une instruction telle que **PLAY**, si la chaîne de caractères est très longue, l'interruption peut n'être prise en compte qu'après plusieurs secondes. Il est nécessaire, dans ce cas, de garder les doigts appuyés sur les touches.

CNT et **X** introduit le caractère **"CONTROLE-X"**. Il efface la fin de la ligne à partir de la position du curseur.

RAZ : Efface le contenu de la fenêtre de travail.

EFF : Touche **"EFFACEMENT"**. Elle supprime le caractère inscrit au-dessus du curseur, la partie de la ligne située à droite de celui-ci étant déplacée d'une position vers la gauche. Cette opération peut être répétée pour supprimer plusieurs caractères.

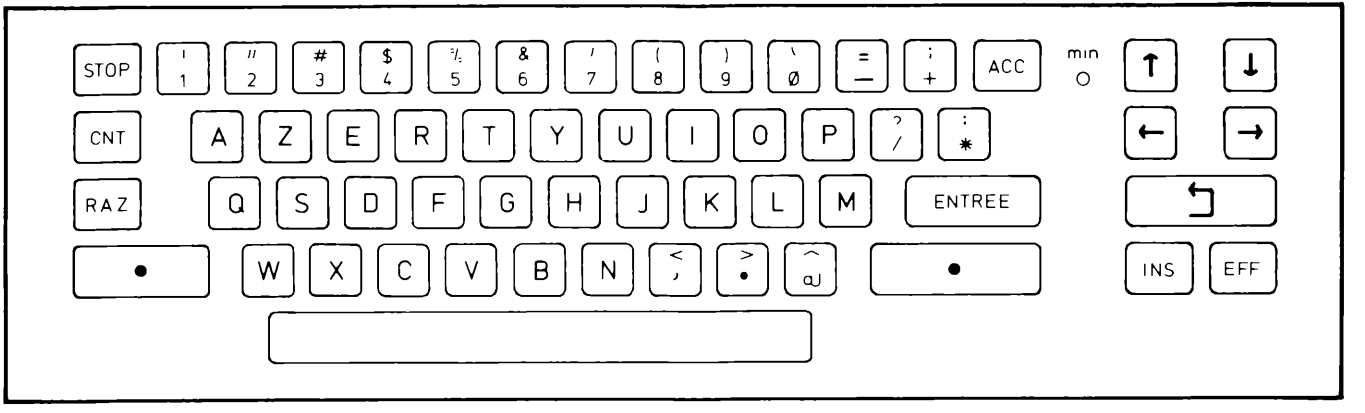
Exemple : **BONNJOUR**
BONJOUR curseur

INS : Touche **"INSERTION"**. Elle crée un espace (blanc) au-dessus du curseur en déplaçant vers la droite la fin de la ligne courante. Cette opération peut être répétée plusieurs fois afin de permettre l'insertion de plusieurs caractères.

Exemple : **BONOUR**
BON OUR curseur

Pour remplacer un caractère :

- Amener le curseur sous le caractère à remplacer.
- Appuyer sur la touche correspondant au caractère désiré.


STOP
1
2
3
4
5
6
7
8
9
Ø
ACC
min
CNT
A
Z
E
R
T
Y
U
I
O
P
RAZ
Q
S
D
F
G
H
J
K
L
M
ENTREE
W
X
C
V
B
N
INS
EFF

I-4 NOTATION UTILISEE

Dans ce manuel, la syntaxe, c'est-à-dire la manière d'écrire les instructions ou fonctions Basic, est décrite en utilisant les règles suivantes :

• Les mots représentant des commandes, instructions, fonctions et d'une manière générale les mots clés du langage sont écrits en majuscules.

Exemple :

```
PRINT "ordinateur personnel"
```

• Tous les signes de **ponctuation** (virgule, point-virgule, tirets, parenthèses, etc.) sont obligatoires à l'exception des caractères < > ou **[ ]**. Dans l'exemple précédent, les guillemets sont indispensables pour éviter une erreur de syntaxe.

• Les expressions en minuscules encadrées par les caractères < > doivent être remplacées par des termes adéquats choisis par l'utilisateur.

Exemple :

GOTO < numéro de ligne >

Cette instruction, insérée dans un programme indique à l'ordinateur qu'il doit effectuer un ''branchement'' au numéro de ligne spécifié par l'utilisateur. Si le numéro de ligne est par exemple 5Ø, l'instruction sera écrite comme suit dans le programme :

• Les termes placés entre crochets sont facultatifs.

Exemple :

LIST [< numéro de ligne >]

Cette instruction permet d'afficher sur l'écran du terminal soit le listage complet du programme soit une ligne de celui-ci selon que l'expression entre crochets est présente ou non Ainsi :

`LIST`	affiche le listage complet du programme
`LIST 35`	affiche seulement la ligne 35 du programme

• Les termes suivis de points de suspension peuvent être répétés plusieurs fois dans les limites de la longueur d'une ligne logique (voir 1-8).

Exemple :

PRINT ''< expression >'' [, ''< expression >'']...

Une ou plusieurs expressions écrites à la suite de **PRINT** seront affichées sur l'écran du terminal.

• Lorsque plusieurs termes sont réunis par des accolades, l'utilisateur doit en choisir un. Deux de ces termes ne peuvent être utilisés simultanément dans une instruction.

Exemple :

GOTO / **GO TO** < **numéro de ligne** >

Les deux syntaxes possibles sont **GOTO** < **numéro de ligne** > et **GO TO** < **numéro de ligne** >.

REMARQUES :

Les espaces entre instructions, termes ou expressions peuvent être omis lors de l'écriture du programme, le Basic de l'ordinateur les interprétant automatiquement. Ainsi **PRINT"ORDINATEUR PERSONNEL"** et **PRINT "ORDINATEUR PERSONNEL"** donnent le même résultat.

Toutefois les espaces dans un mot clé sont interdits car ils se traduiraient par des erreurs de syntaxe. Par exemple l'instruction **PRINT** ne peut s'écrire **P RINT**. Il existe cependant quelques exceptions :

GOTO	peut s'écrire	**GO TO**
GOSUB		**GO SUB**
INPUTWAIT		**INPUT WAIT**
LINEINPUT		**LINE INPUT**

Un nom de variable suivi d'un mot clé doivent être séparés par au moins un espace.

I-5 VALIDATION D'UNE LIGNE

Lors de la frappe d'une ligne, il est possible de corriger d'éventuelles erreurs en utilisant les touches de déplacement du curseur ainsi que les touches d'insertion et effacement. Lorsque la ligne est jugée correcte, une action sur la touche "**ENTREE**" provoque :

- la prise en compte de la totalité de la ligne logique (voir I-8), sous laquelle se trouve le curseur, par l'interpréteur BASIC,
- le passage au début de la ligne suivante.

Le symbole [ENTREE] utilisé dans le chapitre I représente la frappe sur la touche [ENTREE]. Il ne sera plus mentionné dans les chapitres suivants.

I-6 MODES DE FONCTIONNEMENT

Lorsque le message "**OK**" est affiché, l'interpréteur Basic peut travailler selon deux modes différents :

le mode calculateur qui rappelle le fonctionnement d'une calculette,

le mode programme.

I-6-1 Le mode calculateur ou mode commande

Ce mode permet d'effectuer des calculs, d'imprimer la valeur de variables qui ont été évaluées précédemment et d'aider à la mise au point des programmes.

Il consiste essentiellement à introduire des instructions sans numéro de ligne, qui sont exécutées immédiatement. La plupart des instructions (sauf par exemple **INPUT**) peuvent être utilisées de même que les commandes suivantes : **AUTO, CONT, DELETE, LIST, LOAD, MERGE, NEW, RUN** et **SAVE**.

Exemple :

```
PRINT 345.12*1.176          ENTREE
 4Ø5.861
OK
HT=12ØØØ                    ENTREE
OK
TTC=HT*1.175                ENTREE
OK
PRINT TTC                   ENTREE
 14112
OK
```

I-6-2 Le mode programme

Dans ce mode, les instructions sont précédées d'un numéro de ligne choisi dans la plage **Ø - 63999**. Les instructions ainsi introduites sont stockées en mémoire centrale et ne sont exécutées que si on frappe la commande **RUN**.

Un programme peut contenir des commandes. L'exécution de certaines d'entre elles (**LIST, NEW** etc) interrompt le déroulement du programme en cours et provoque le retour au mode commande

Introduction et modification d'un programme

La constitution d'un programme nouveau peut se faire de différentes manières .

- en frappant le numéro de ligne, l'instruction et la touche **ENTREE**,
- en utilisant la commande **AUTO** (numérotation automatique) qui évite de frapper les numéros de ligne, puis frapper l'instruction et la touche **ENTREE**,
- en utilisant la commande **MERGE** qui permet de fusionner les instructions en mémoire avec une séquence d'instructions déjà stockée sur mémoire auxiliaire.

Il est également possible de modifier des lignes de programme déjà introduites en utilisant l'éditeur ''plein écran''.

- **remplacement d'une ligne** :
  Frapper à nouveau et entièrement la ligne concernée, la nouvelle ligne remplaçant l'ancienne.
- **modification d'une ligne** :
  Placer le curseur sur la ligne à modifier puis effectuer les modifications désirées. Dès que la ligne à modifier est jugée correcte, l'action sur la touche **ENTRÉE** la fait prendre en compte par l'ordinateur.
- **suppression d'une ligne** :
  Il suffit de frapper son numéro et la touche **ENTREE** ou d'utiliser l'instruction **DELETE** (cf. chapitre II).
- **insertion d'une nouvelle ligne** :
  Il suffit de frapper le numéro que l'on veut lui donner, puis le contenu de la ligne et la touche **ENTREE**.

Conseil : numéroter les lignes avec un incrément suffisant, car la commande de renumérotation n'existe qu'avec la version disque. Un incrément de 1Ø est en général suffisant.

I-7 TAILLE ET COULEUR DES CARACTERES

Le Basic de l'ordinateur **TO7** permet d'afficher sur l'écran, des caractères de différentes tailles : hauteur normale ou double hauteur, largeur normale ou double largeur. Cela permet de déterminer 4 tailles possibles pour les caractères : normale, double largeur, double hauteur, double taille (hauteur et largeur doubles).

Le choix de la taille des caractères se fait au moyen de l'instruction **ATTRB**. Cette instruction permet en outre de travailler en ''mode masqué'' qui rend invisibles les caractères affichés sur l'écran, (caractères noirs sur fond noir). Les caractères peuvent être rendus visibles au moyen de l'instruction **UNMASK**.

Le retour en mode commande (fin d'exécution ou interruption du programme) fait revenir à des caractères de taille normale et au mode non masqué.

Une instruction telle que **COLOR** permet de choisir la couleur des caractères ainsi que celle du fond sur lequel ils seront affichés parmi les couleurs suivantes : noir, rouge, vert, jaune, bleu, magenta (violet), cyan (bleu clair) et blanc.

I-8 FORME DES LIGNES

Les lignes d'un programme BASIC doivent avoir la forme générale suivante :

< **numéro de ligne** > < **instruction BASIC** > [: < **instruction BASIC** >]...

L'utilisateur peut à son gré, introduire une ou plusieurs instructions par ligne, séparées les unes des autres par le caractère :

Les lignes sur l'écran comportent au maximum 4Ø caractères. Il est toutefois possible d'introduire des lignes de programme allant jusqu'à 255 caractères. Nous parlerons donc de ligne physique et de ligne logique : une ligne physique est une ligne de l'écran, une ligne logique est une ligne de programme ou de données et peut occuper plusieurs lignes physiques

Lors de l'introduction d'une ligne logique comportant plus de 4Ø caractères, le passage à la ligne physique suivante est commandé par l'interpréteur Basic Il ne faut frapper la touche **ENTREE** qu'à la fin de la ligne logique.

Les espaces sont facultatifs sauf quand une variable précède un mot clé. On peut insérer autant d'espaces que l'on veut. Les espaces sont interdits à l'intérieur d'une constante numérique, d'une variable ou d'un mot clé.

Les mots clés et les variables peuvent être écrits en majuscules ou en minuscules non accentuées, l'interpréteur Basic transformant automatiquement les minuscules en majuscules.

Exemple :

```
For i=1 to xmin
```

est équivalent à

```
FOR I=1 TO XMIN
```

I-9 ALPHABET UTILISE

L'ordinateur **TO7** utilise le jeu de caractères ASCII. Il a été étendu aux minuscules accentuées et au ç.

Pour plus de détails, il est conseillé de lire l'annexe E.

L'utilisateur peut définir des graphismes de code 128 à 255 au moyen de l'instruction **DEFGR$** et les utiliser ensuite comme des caractères normaux, à ceci près qu'ils s'appelleront **GR$(Ø)**, **GR$(1)** etc. jusqu'à **GR$(127)**. Le chapitre V contient un exemple de programme créant et affichant de tels graphismes.

RAPPEL DES NOTIONS FONDAMENTALES DU LANGAGE BASIC

Les utilisateurs dont les connaissances du langage BASIC sont suffisantes peuvent sauter les paragraphes suivants.

I-10 CONSTANTES

I-10-1 Type des constantes

Le Basic accepte des constantes numériques et des constantes "chaîne".

Les constantes numériques peuvent être de type :
- entier
- réel ou simple précision
- double précision

Il accepte en outre des constantes représentées sous forme hexadécimale, octale ou binaire.

I 10-2 Constantes numériques

Une constante entière est représentée par un nombre entier suivi du caractère %.

- Une constante réelle est représentée par un nombre avec ou sans partie fractionnaire et avec ou sans exposant. L'exposant est précédé de la lettre **E**. S'il n'y a pas d'exposant, un point d'exclamation (facultatif) ne peut être ajouté après la constante afin de préciser sa nature.

- Une constante double précision est représentée de la même façon qu'une constante réelle à ceci près que l'exposant est précédé de la lettre **D** au lieu du **E**. Si il n'y a pas d'exposant, ce nombre doit être suivi du caractère #.

Exemples :

12%
−1234%
+12%
} sont des constantes entières

12
1234!
−1.6E19
1.6E−19
} sont des constantes réelles

12#
3.14159267#
−.314159267D+15
} sont des constantes double précision

La plage de variation des nombres est la suivante :

- entier : de −32768 à +32767
- réel : l'exposant peut varier de −38 à +38 et la mantisse ne peut pas avoir plus de 7 chiffres significatifs (dans la pratique la représentation interne donne une précision entre 6 et 7 chiffres décimaux).
- double précision : même plage de variation pour l'exposant, mais la mantisse peut avoir jusqu'à 16 chiffres significatifs

I-10-3 Constantes chaîne

Une constante chaîne est une suite de caractères encadrée par les caractères guillemets.

Exemple :

"BELLE MARQUISE"
"Bonjour, comment allez-vous ?"

Une constante chaîne de même qu'une variable chaîne peuvent comporter de Ø à 255 caractères.

Une chaîne ne comportant aucun caractère est dite chaîne vide.

La longueur d'une chaîne est définie par le nombre de caractères qu'elle contient.

Les minuscules accentuées ou le c cédille nécessitent la frappe de trois caractères. Il y a lieu d'en tenir compte dans le calcul de la longueur d'une chaîne.

I 10-4 Constantes hexadécimales et octales

Une constante hexadécimale est un nombre en base 16 précédé des caractères **&H**.

Une constante octale est un nombre en base 8 précédé des caractères **&O** ou **&**.

Exemples :

&H76, &H32F } sont des constantes hexadécimales

&0347, &1234 } sont des constantes octales

Ces constantes peuvent être affectées à des variables de type quelconque.

I 10-5 Constantes binaires

Une constante binaire est une séquence de Ø à 16 chiffres binaires (Ø ou 1), précédée de la séquence **&B**.

Exemples :

&BØØØØØØØ11ØØ11111 ou plus simplement &B11ØØ11111
&BØØØØØØØØØØØØØØØ1 ou simplement &B1

Les valeurs de ces constantes peuvent être affectées à des variables (en particulier à des variables entières) et permettent des opérations booléennes au moyen des opérateurs **AND**, **EQV**, **IMP**, **NOT**, **OR** et **XOR**.

I 11 VARIABLES

I 11-1 Noms de variables

Les noms de variables peuvent avoir une longueur de 1 à 4Ø caractères, le premier étant une lettre, les autres pouvant être des lettres ou des chiffres.

Toutefois seuls, les 16 premiers caractères sont pris en compte : deux variables dont les seize premiers caractères sont identiques sont, pour le BASIC, identiques.

Exemples :

A, A3B } sont des noms de variables corrects

3AB n'est pas un nom de variable

Un nom de variables ne doit ni être un mot réservé, ni commencer par un mot réservé du langage. Voir en annexe D.

Par exemple **TOTAL, VALEUR** ne peuvent pas être des noms de variables car **TOTAL** commence par **TO** qui entre dans l'instruction **FOR...TO** et **VAL** est un nom de fonction.

I-11-2 Type de variables

Le Basic accepte quatre types de variables :

- variables entières,
- variables simple précision ou ''réelles'' (c'est le cas habituel),
- variables ''double précision'',
- variables ''chaîne''.

Le type des variables peut être indiqué de deux manières :

- en faisant suivre le nom de la variable du caractère :
  - % pour les variables entières
  - # pour les variables ''double précision''
  - $ pour les variables ''chaîne''
  - ! ou rien pour les variables ''réelles''
- en utilisant une instruction de déclaration de type telle que :
  DEFINT, DEFSNG, DEFDBL et **DEFSTR**

Ces instructions sont expliquées dans le chapitre II.

Une variable dont le type n'est pas indiqué est considérée comme étant de type réel.

Exemples :

TAUX	est de type réel
T!	est de type réel
CENT%	est de type entier
X#	est de type double précision
A$	est de type chaîne

I-11-3 Tableaux

Un tableau est un ensemble de variables de même type, référencé par un nom unique.

Chaque élément est désigné par ce nom suivi par un ou plusieurs indices.

On parlera alors de variables indicées

Il faut autant d'indices qu'il y a de dimensions au tableau.

Par exemple A(4) désigne un élément d'un tableau A à une seule dimension, B(1,4) un élément d'un tableau B à deux dimensions et ainsi de suite

Exemples de tableau :

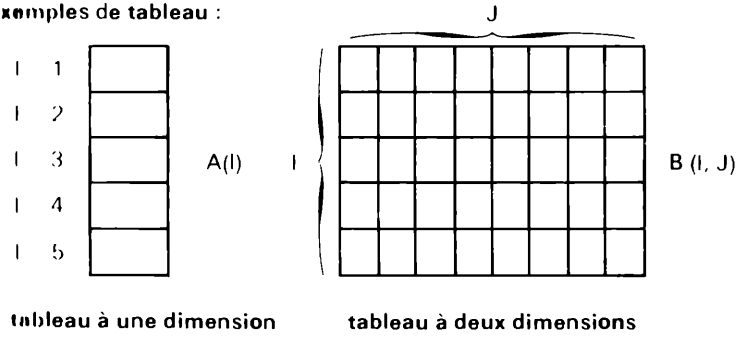


tableau à une dimension **tableau à deux dimensions**

La syntaxe du BASIC accepte des tableaux ayant de 1 à 255 indices, chaque indice pouvant prendre des valeurs de Ø à 32767.

Il s'agit là de limites théoriques car l'utilisateur sera limité par l'espace mémoire disponible

Pour les variables ayant 1 ou 2 indices, et si la valeur maximum des indices n'excède pas 1Ø, la déclaration n'est pas obligatoire mais recommandée. Les autres variables indicées doivent être déclarées au moyen de l'instruction **DIM**.

I 11 4 Indices

Si l'évaluation d'un indice conduit à une valeur non entière, la valeur est convertie en entier par arrondi (et non pas par troncature) pour donner ensuite l'élément de tableau.

Exemples :

A(3.5) est équivalent à A(4)
A(3.49) est équivalent à A(3)

Un indice peut être du type entier ou réel mais pas du type double précision ou chaîne. Les indices commencent à Ø.

I 12 CONVERSIONS

Le Basic accepte des expressions "mixtes" c'est-à-dire faisant intervenir des variables ou des constantes numériques de type différent. Par contre, il n'accepte pas qu'un opérateur porte sur une valeur de type chaîne et une valeur de type numérique.

Exemples :

X + 3*LEN(A$)	est valable
X + A$	est incorrect

L'évaluation d'une expression se fait en convertissant s'il y a lieu les diverses valeurs numériques au type de précision de la valeur la plus précise ; le sens de la précision croissante est : entier, réel puis double précision. L'évaluation de fonctions mathématiques standards se fait cependant en simple précision.

Exemples :

X = A%*Z	La valeur de A% est convertie en réel avant d'effectuer la multiplication
X = A%*Z + D#	A% est converti en réel, puis il y a multiplication et le résultat est à son tour converti en double précision avant l'addition.

S'il y a lieu, la valeur de l'expression est ensuite convertie dans le type de la variable à affecter. Ainsi dans l'exemple précédent, le résultat de l'expression est converti en réel avant affectation à la variable X.

Cette conversion est faite avec arrondi.

Exemple :

A% = 3.25	A% reçoit la valeur 3
A% = 3.5	A% reçoit la valeur 4

L'affectation d'une valeur double précision à une variable réelle se fait en gardant les 7 premiers chiffres significatifs.

I-13 EXPRESSIONS ET OPERATEURS

Une expression peut être une constante, une variable, ou encore une combinaison de constantes, variables et fonctions reliées entre elles par des opérateurs et des parenthèses.

Les opérateurs peuvent être divisés en 4 catégories :

- opérateurs arithmétiques
- opérateurs de relations
- opérateurs logiques.
- opérateurs sur chaînes.

I 13 1 Opérateurs arithmétiques

Le tableau suivant contient les opérateurs par priorité décroissante :

↑	élévation à la puissance	**X↑ Y**
–	moins unaire	**– X**
*, /	multiplication et division	**X*Y et X/Y**
@	division entière	**X @ Y**
MOD	modulo	**X MOD Y**
+ , –	addition et soustraction	**X + Y – Z**

Le caractère ↑ est obtenu en appuyant simultanément sur les touches [·] et [@]

•

Opérateurs division entière et modulo

Ces opérateurs permettent d'effectuer des opérations utiles en arithmétique :

- la division entière
- le reste de la division entière.

Les deux opérandes sont d'abord évalués et leur valeur est, si nécessaire, convertie en entier par arrondi. Le signe du résultat est celui du premier opérande. L'opération est ensuite effectuée :

- division entière notée @ donne le quotient de la division entière.
- modulo noté **MOD** donne le reste de la division entière.

Exemples :

10.3 @ 2	donne 5	
10.3 @ 3	donne 3	
10.3 @ 3.5	donne 2	(correspond à 10 ÷ 4)
10 MOD 2	donne 0	
10 MOD 3	donne 1	
10.49 MOD 3.1	donne 1	(correspond à 10 MOD 3)
10.5 MOD 3.49	donne 2	(correspond à 11 MOD 3)

Dépassement de capacité et division par zéro

Tout calcul conduisant à un dépassement de capacité par valeur **supérieure** (overflow en anglais) est signalé par le message :

OV Error suivi du numéro de la ligne où l'erreur a été détectée

Une division par zéro est signalée par le message :

/0 Error suivi du numéro de ligne où l'erreur a été détectée

Dans chacun de ces deux cas, l'exécution est interrompue.

I-13-2 Expressions numériques

- Les opérations sont effectuées dans l'ordre des priorités défini en I-13-1.
- Les fonctions sont prioritaires par rapport à l'élévation à la puissance.
- Si l'expression contient des parenthèses, les opérations situées à l'intérieur de celles-ci sont effectuées en premier en respectant l'ordre des priorités.

I-13-3 Opérateurs de relation

Les opérateurs de relation sont utilisés pour comparer deux valeurs. Le résultat de la comparaison est une valeur "booléenne" **VRAI** ou **FAUX**. Par convention, **"VRAI"** a pour valeur -1 et **"FAUX"** a pour valeur Ø.

Ces résultats peuvent être utilisés pour les instructions de branchement.

OPERATEUR	RELATION TESTEE	EXPRESSION
=	égalité	X = Y
<>	inégalité	X <> Y
<	inférieur à	X < Y
>	supérieur à	X > Y
<=	inférieur ou égal à	X <= Y
>=	supérieur ou égal à	X >= Y

Notons que le caractère $=^{\bullet}$ est également utilisé comme symbole d'affectation.

Quand des opérateurs de relation et des opérateurs arithmétiques apparaissent au sein d'une même expression, les opérations arithmétiques sont d'abord effectuées. Par exemple :

```
X + Y < (T + I) / Z
```

est vraie si (X + Y) est inférieur à (T + I) / Z

Le résultat d'une telle comparaison peut être affecté à une variable en vue d'être exploité ultérieurement. On peut par exemple écrire :

```
B = X + Y < (T + I) / Z
```

I-13-4 Opérateurs logiques (ou booléens)

Ces opérateurs permettent d'effectuer des opérations booléennes et des comparaisons sur plusieurs relations.

Six opérateurs logiques sont disponibles :

AND	ET logique
EQV	équivalence
IMP	implication
NOT	complément
OR	OU logique (inclusif)
XOR	OU exclusif

NOT est un opérateur "unaire" : il ne s'applique qu'à un seul opérande. Si A est "vrai" et B est "faux" :

NOT (A) donne Ø (FAUX)
NOT (B) donne – 1 (VRAI)

Les autres opérateurs dont la table de vérité est donnée ci-dessous sont des opérateurs binaires (deux opérandes).

1er opérande	2e opérande	X AND Y	X EQV Y	X IMP Y	X OR Y	X XOR Y
Ø	Ø	Ø	1	1	Ø	Ø
Ø	1	Ø	Ø	1	1	1
1	Ø	Ø	Ø	Ø	1	1
1	1	1	1	1	1	Ø

Les opérateurs logiques s'appliquent à des opérandes de 16 bits. Les opérandes réels ou double précision sont convertis en représentation entière et leur valeur doit donc se trouver dans la plage – 32768 à + 32767.

Dans les mots de 16 bits, les opérations sont effectuées sur les bits de même poids.

Exemples :

123 AND &B1111111111111110 permet de supprimer le bit de poids faible d'un nombre entier pour obtenir un résultat pair.

Pour savoir si un entier X% est pair, on peut par exemple écrire :

IF X% AND &B1111111111111111Ø = X% THEN

Pour obtenir le plus grand nombre pair inférieur ou égal à X%, on pourra écrire :

X% AND (– 2)

Pour obtenir le plus petit nombre impair supérieur ou égal à X%, on pourra écrire :

X% OR (1)

I-13-5 Opérations sur chaîne de caractères

Le seul opérateur portant sur des chaînes est l'opérateur de concaténation noté + . Considérons, par exemple, le programme suivant :

```
10 A$="VILLE"          ENTREE
20 B$="VIEILLE"        ENTREE
30 C$=" VIEILLE"       ENTREE
40 R1$=A$+B$           ENTREE
50 R2$=A$+C$           ENTREE
60 PRINT R1$,R2$       ENTREE
```

L'exécution de ce programme donnera le résultat suivant :

```
RUN
VILLEVIEILLE VILLE VIEILLE
OK
```

La concaténation de chaîne se fait en mettant les chaînes bout à bout sans insérer de caractères "espace".

Les chaînes peuvent être comparées au moyen des opérateurs de relation comme les expressions numériques. La comparaison se fait caractère par caractère à partir du début de chaque chaîne. Un caractère est dit plus petit qu'un autre s'il a un plus petit code dans le jeu de caractères ASCII (Cf. annexe E).

Tant que l'ordinateur trouve des caractères égaux, il poursuit la comparaison. Dès qu'il trouve un caractère de l'une des chaînes plus petit que le caractère correspondant de l'autre chaîne, la première chaîne est considérée comme plus petite que l'autre.

Si la fin d'une chaîne est atteinte avant qu'une différence ne soit apparue, alors la chaîne la plus courte est considérée comme la plus petite.

Dans cette comparaison, tous les caractères sont pris en compte y compris les espaces éventuels en début et en fin de chaîne.

Exemples :

"AB"	< "FG"	car "A" < "F"
"ABC"	< "ABCD"	car la première chaîne est plus courte
"TUER"	> "TRETEAU"	car "U" > "R"
"CHAT"	< "CHAt"	car "T" < "t"

Les expressions à valeur chaîne c'est-à-dire toute suite de caractères pouvant comprendre des caractères alphabétiques, chiffres, signes de ponctuation ou spéciaux, peuvent être conçues en utilisant, outre l'opérateur de concaténation, des fonctions "chaîne" présentées au chapitre III.

I.14 DETECTION ET GESTION DES ERREURS

Si le programme est écrit sans précaution particulière, toute erreur détectée par l'interpréteur BASIC donne lieu à un message d'erreur caractérisé par deux lettres suivies du mot **Error.**

Par contre, un programme peut contenir (en général en tête du programme) une instruction **ON ERROR** qui indique la ligne de branchement dans le cas où une erreur est détectée.

A partir de cette ligne il peut y avoir une séquence d'instructions qui traite cette erreur. Par l'intermédiaire des variables **ERR** et **ERL** il est possible de connaître le type de l'erreur **(ERR)** et le numéro de ligne où celle-ci s'est produite **(ERL)**. L'annexe C donne la liste des numéros d'erreurs.

Exemple :

```
10 ON ERROR GOTO 60
                'aller en 60 en cas
                 d'erreur                    ENTREE
20 INPUT X                                   ENTREE
30 A=10/X                                    ENTREE
40 PRINT A      'afficher sur l'écran        ENTREE
50 GOTO 20                                   ENTREE
60 IF ERR=11 THEN PRINT
"donner X différent de zéro"
                'si l'erreur est div/0
                 afficher le message         ENTREE
70 RESUME 20    'continuer en 20             ENTREE

RUN
? 4             ENTREE
 2.5
? 1             ENTREE
 10
? 0             ENTREE
donner X différent de zéro
? -3            ENTREE
-3.33333
?          CNT      et      C
Break In 20
OK
```

I-15 PARAMETRES IMPLICITES

En général, lorsqu'un paramètre facultatif est absent dans une instruction, sa valeur (définie précédemment) n'est pas modifiée.

Exemple :

```
SCREEN   ,1
```

La couleur de fond prend la valeur 1 (rouge), mais les couleurs du graphisme et de l'entourage sont inchangées (1er et 3e paramètres absents).

II - COMMANDES ET INSTRUCTIONS

Le chapitre II décrit l'ensemble des commandes et instructions de l'interpréteur de la cartouche BASIC à l'exception des instructions de fichiers et de graphisme traitées dans les chapitres IV et V. Les commandes et instructions sont décrites par ordre alphabétique selon le modèle suivant :

NOM de la commande ou de l'instruction

Syntaxe : donne la ou les syntaxes possibles selon la notation décrite au chapitre I, paragraphe 1-4.

But indique de façon détaillée le rôle de la commande ou de l'instruction sur le plan sémantique en particulier.

Exemple court programme ou portion de programme à titre d'illustration.

Remarque paragraphe non systématique qui donne des informations complémentaires sur l'utilisation, la sémantique ou le lien avec d'autres instructions, commandes ou fonctions.

Conseil paragraphe non systématique donnant des conseils de mise en œuvre.

Le lecteur trouvera dans les chapitres IV et V des exemples de mise en œuvre de certaines instructions et fonctions.

ATTRB

Syntaxe

ATTRB [L] [,[H][,M]]

où **L** = largeur des caractères, **H** = hauteur des caractères, et **M** = mode d'affichage.

But :

Cette instruction permet de définir la taille des caractères qui seront affichés ensuite sur l'écran. Elle permet en outre de spécifier le mode normal ou le mode masqué.

Les trois paramètres sont des expressions dont les valeurs après conversion éventuelle en entier ont la signification suivante :

- **L** définit la largeur des caractères
  - **L = Ø** largeur normale
  - **L = 1** largeur double
- **H** définit la hauteur des caractères
  - **H = Ø** hauteur normale
  - **H = 1** hauteur double
- **M** permet de spécifier le mode d'affichage
  - **M = Ø** mode normal
  - **M = 1** mode masqué

En l'absence d'un paramètre, le mode correspondant en vigueur précédemment n'est pas changé.

Le mode "masqué" entraîne sur l'écran l'affichage de caractères noirs sur fond noir. Ces caractères masqués ne sont pas perdus et pourront être "démasqués" au moyen de l'instruction **UNMASK.**

Le mode "masqué" disparaît lorsque l'ordinateur revient en mode commande mais les caractères masqués le restent jusqu'à ce qu'une instruction **UNMASK** ou **SCREEN** les rende visibles.

En mode double hauteur, les instructions **PRINT** provoquent l'affichage sur la ligne courante et sur la ligne précédente. En double hauteur, un passage à la ligne provoque un double interligne. L'affichage de caractères agrandis est possible dans les conditions suivantes :

- caractères **ACSII**,
- graphismes utilisateurs (Cf. **DEFGR$**),
- caractères normaux **TELETEL**.

Les caractères semi-graphiques **TELETEL** ne peuvent pas être affichés en mode agrandi.

L'instruction **ATTRB** n'a d'effet ni sur les sorties vers imprimante ou fichiers ni sur les caractères semi-graphiques **TELETEL**.

L'exécution d'une instruction **INPUT** ou **LINE INPUT**, après affichage éventuel du texte associé, provoque le passage en mode normal. Les fonctions **INPUT$** et **INKEY$** ne changent pas le mode d'affichage

Exemple :

```
WIDTH 1,1
```
fera afficher sur l'écran des caractères en double largeur et double hauteur

```
TYPE ,1
```
fera passer en double hauteur sans changer la largeur des caractères.

Remarques :

- Quand l'exécution du programme se termine, il y a retour automatique à des caractères de taille normale avec un mode non masqué.
- En mode double hauteur, pour le premier affichage, afin d'éviter d'écrire sur la ligne précédente, il faut prévoir l'envoi préalable d'une ligne vide au moyen d'une simple instruction **PRINT**. Par exemple, on pourra écrire :

On peut aussi remplacer l'instruction de la ligne 11Ø par une instruction **LOCATE** pour positionner le curseur

Ensuite, en mode double hauteur, il y a automatiquement double interligne.

Conseils :

L'utilisateur peut utiliser le mode masqué :

pour demander un ''mot de passe'' qui reste ainsi invisible.

- pour cacher un texte ou un graphisme qui sera ensuite dévoilé d'un seul coup au moyen de l'instruction **UNMASK**.

AUTO

Syntaxe

AUTO [< numéro de ligne >] [, [< incrément >]]

où < **numéro de ligne** > et < **incrément** > sont des constantes entières positives.

But :

Cette commande permet de faire afficher par l'ordinateur des numéros de ligne croissant de façon automatique.

Les numéros de ligne produits commencent au numéro figurant dans la commande. Chaque appui sur la touche **ENTREE** provoque l'affichage du numéro suivant dont la valeur est celle du précédent augmenté de l'incrément.

Si < **numéro de ligne** > et < **incrément** > sont omis, la valeur par défaut 1Ø est prise.

Si le < **numéro de ligne** > est suivi d'une virgule mais sans < **incrément** > alors l'incrément utilisé est celui en vigueur lors de la numérotation automatique précédente.

Si le < **numéro de ligne** > est omis, la valeur par défaut est 1Ø.

Si cette numérotation automatique produit un numéro de ligne déjà utilisé, un astérisque est affiché juste après le numéro, afin de prévenir l'utilisateur que l'introduction de cette ligne remplacera le texte antérieur portant le même numéro. Si l'utilisateur désire poursuivre la numérotation automatique sans détruire la ligne existante, il devra appuyer sur la touche "**ENTREE**" juste après l'astérisque. Un nouveau numéro de ligne sera alors produit selon la règle indiquée précédemment.

Pour arrêter la numérotation automatique il faut introduire le caractère "**CNT C**", la ligne en cours d'introduction est alors perdue et l'interpréteur revient en mode commande.

La numérotation automatique peut également être interrompue par déplacement du curseur en appuyant sur la touche "↑" ou "↓" et ensuite sur la touche "**ENTREE**".

Exemples :

AUTO 1Ø,5	*produit la séquence de numéros 1Ø, 15, 2Ø, etc.*
AUTO	*produit la séquence de numéros 1Ø, 2Ø, 3Ø, etc.*
AUTO 15	*produit la séquence de numéros 15, 25, 35, etc.*
AUTO ,5	*produit la séquence de numéros 1Ø, 15, 2Ø, etc.*

Si on introduit d'abord la commande

AUTO 1Ø,2Ø *la séquence 1Ø, 3Ø, 5Ø, etc. est produite*

Si l'utilisateur frappe simultanément sur les touches **CNT** et **C** - puis

AUTO 2Ø, *la séquence 2Ø, 4Ø, 6Ø, etc sera produite.*

BEEP

Syntaxe :

BEEP

But :

Cette instruction produit un son court. Elle est équivalente à **PRINT CHR$(7)** car, dans le code ASCII, le caractère de code 7 représente la sonnette.

Exemple :

```
BEEP
```

Remarques :

1 - Pour obtenir un son plus long, il suffit d'insérer **BEEP** dans une boucle **FOR** ainsi que le montre l'exemple suivant :

```
100 FOR I= 1 TO 30
110 BEEP
120 NEXT
```

2 - Il est possible d'obtenir des sons plus longs et de fréquence différente au moyen de l'instruction **PLAY**.

3 - Dans un ordre **PRINT, CHR$ (7)** tient la place de 1 caractère ; **BEEP** n'occupe pas de place...

BOX

Voir chapitre V.

BOXF

Voir chapitre V.

CLEAR

Syntaxe :

CLEAR [I] [, [J] [, K]]

I, **J** et **K** sont des expressions dont les valeurs sont, si nécessaire, arrondies à l'entier le plus proche. Elles ne doivent pas être négatives. La valeur de **K** ne doit pas excéder 128.

I = réservation chaîne ; **J** = réservation mémoire ; **K** = réservation caractères utilisateurs.

But :

Cette instruction réalise plusieurs fonctions :

- elle détruit et récupère la place des variables simples et tableaux qui auraient été créés avant son exécution.
- elle annule l'effet d'une instruction **ON ERROR GOTO**.
- **I** permet de réserver de la place pour la manipulation de chaînes.

  Quand on manipule plusieurs chaînes, il faut disposer de la place nécessaire pour les faire toutes figurer en même temps.

  Par exemple, l'instruction : **C$ = C$ + B$** nécessite une place correspondant **B$**, **C$** et à la chaîne obtenue par concaténation.

  Dès que le programme a besoin d'une place supérieure à 3ØØ caractères, cette place doit être réservée au moyen de l'instruction **CLEAR**.

- **J** indique, s'il est présent, l'adresse maximum que peut utiliser l'interpréteur Basic. Cela permet ainsi de réserver de la place pour des sous-programmes écrits en ''langage machine''. En son absence, l'interpréteur Basic utilise tout l'espace mémoire disponible.

- **K** indique le nombre de caractères graphiques que veut créer l'utilisateur de façon que l'interpréteur réserve la place en conséquence. L'instruction **CLEAR** reste en effet jusqu'au prochain **CLEAR**, même après une réinitialisation.

Exemple 1 :

Concaténation de chaînes longues sans réservation préalable.

```
1Ø A$="12345678Ø1234567ØØ" 'longueur=2Ø
2Ø FOR I=1 TO 12
3Ø B$=B$+A$                'B$ doit atteindre
                            la longueur 24Ø
4Ø NEXT I

RUN
?OS Error In 3Ø
```

Exemple 2 :

Concaténation après réservation au moyen de l'instruction **CLEAR**.

```
5 CLEAR5Ø9
1Ø A$="12345678Ø1234567Ø"
2Ø FOR I=1 TO 12
3Ø B$=B$+A$
4Ø NEXT I                          'il ne se produit
                                    pas d'erreur

RUN
OK
```

Exemple 3 :

Réservation de place pour introduire 3 caractères graphiques utilisateur.

```
1Ø CLEAR ,, 3
2Ø DEFGR$(Ø)= .....
3Ø DEFGR$(1)= .....
4Ø DEFGR$(2)= .....
```

Exemple 4 :

Combinaison des cas précédents : 6ØØ octets pour les chaînes, 2 caractères graphiques et l'interpréteur Basic ne doit pas utiliser des adresses au-delà de 34ØØØ.

```
1Ø CLEAR 6ØØ,34ØØØ,2
```

Exemple 5 :

```
1Ø DIM A(3Ø,3Ø)

.................

'ØØ CLEAR          'récupérer la place prise
                    par toutes les variables
```

CLOSE

Voir chapitre IV.

CLS

Syntaxe :

CLS

But :

Cette instruction efface le contenu de la fenêtre de travail et positionne le curseur dans le coin supérieur gauche de la fenêtre.

Exemple :

```
CLS
```

Remarques :

1 - Cette instruction donne le même résultat que **PRINT CHR$(12)** ;
2 - Elle ne modifie pas le choix des couleurs qui aurait été éventuellement fait préalablement à son exécution.

Conseil :

Pour des informations détaillées concernant la fenêtre de travail, lire le texte relatif à l'instruction **CONSOLE**.

COLOR

Voir chapitre V.

CONSOLE

Syntaxe :

CONSOLE [A] [,[B] [,[I] [, J]]]

Deux restrictions à cette syntaxe :

cette instruction ne doit pas se terminer par une virgule,

elle doit comporter au moins un paramètre.

A = ligne supérieure fenêtre ; **B** = ligne inférieure fenêtre ; **I** = blocage couleur ; **J** = mode fenêtre.

But :

Cette instruction permet de définir une fenêtre de travail.

L'expression **A** indique la ligne supérieure de la fenêtre.

L'expression **B** indique la dernière ligne de la fenêtre.

Les valeurs de **A** et **B** après arrondi éventuel, doivent être comprises entre Ø et 24.

La valeur de l'expression **A** ne doit pas être supérieure à celle de l'expression **B**, sinon l'erreur **FC** est détectée.

L'expression **I** doit prendre la valeur Ø ou 1.

Le système TO7 écrit en même temps les points "fond" et "graphique" ainsi que la paire de couleur afférente à chaque segment (Cf. chapitre V).

Lorsque **I = Ø** tous les affichages rencontrés après l'exécution de l'instruction se feront dans la couleur courante.

Lorsque **I = 1** les couleurs prises pour l'affichage seront celles de la ligne de l'écran sur laquelle l'affichage sera réalisé.

Une instruction **COLOR** affectera l'attribut général des couleurs mais son effet ne sera visible que lorsque I = Ø réinitialisera à nouveau l'écriture des couleurs.

— L'expression **J** doit avoir pour valeur (après arrondi éventuel) l'une des trois valeurs Ø, 1 ou 2.

- **Ø** indique que l'ordinateur doit utiliser la fenêtre en mode défilement à vitesse normale
- **1** indique que l'ordinateur doit utiliser la fenêtre en mode défilement doux
- **2** indique que la fenêtre doit être utilisée en mode page.

Le retour en mode commande fait quitter le mode page et revenir au mode défilement.

Pour sortir de la fenêtre de travail, il faut exécuter une instruction **CONSOLE** définissant une autre fenêtre, ou exécuter une instruction **LOCATE** qui déplace le curseur.

Cette instruction est sans effet sur les instructions graphiques.

Exemples :

`CONSOLE Ø,24`	*tout l'écran est utilisé*
`CONSOLE 1Ø,2Ø`	*seules les lignes 1Ø à 2Ø sont utilisées*
`CONSOLE 1Ø,2Ø,1`	*les couleurs ne bougent pas*
`CONSOLE 1Ø,2Ø, ,2`	*les lignes 1Ø à 2Ø sont utilisées en mode page.*

Nota :

•

Dans les modes défilement normal et doux, tout se passe comme si l'écran visualisait une partie d'un rouleau de papier qui défilerait vers le haut ou vers le bas.

Dans le mode ''page'', lorsqu'une page est pleine, la ligne suivante apparaît sur la première ligne de la page.

CONT

Syntaxe :

CONT

But :

Cette commande provoque la reprise de l'exécution d'un programme qui vient d'être interrompue par l'envoi du caractère [CNT]-[C] ou après l'exécution d'une instruction **STOP** ou **END**.

L'exécution reprend à l'endroit où elle a été interrompue. Si [CNT]-[C] a été envoyé au cours de la réponse à une instruction **INPUT**, à la reprise le texte éventuel spécifié dans **INPUT** apparaît à nouveau, ou s'il n'y a pas de texte à éditer, le point d'interrogation est à nouveau affiché à l'écran.

La commande **CONT** est utile pour la mise au point du programme en liaison avec l'instruction **STOP**. Quand l'exécution est suspendue, le contenu des variables peut être affiché ou imprimé au moyen d'une instruction **PRINT** en mode immédiat et même modifié par une affectation. La commande **CONT** permet de reprendre l'exécution à l'endroit même où elle s'était arrêtée à condition de ne pas avoir modifié le programme après l'interruption.

Exemple :

```
CONT
```

Remarque :

En mode immédiat, l'instruction **GOTO** permet de poursuivre l'exécution à un numéro de ligne déterminé.

DATA

Syntaxe :

DATA < **constante** > **[,** < **constante** >**]** ...

But :

Cette instruction permet d'incorporer au programme une liste de données sous forme de constantes qui pourront être lues au moyen d'instructions **READ**.

Les instructions **DATA** ne sont pas exécutables et peuvent se trouver n'importe où dans le programme. L'instruction **DATA** peut contenir autant de constantes que la longueur de la ligne le permet et il n'y a pas de limite au nombre des instructions **DATA** contenues dans un programme.

Les constantes de la liste peuvent être de type entier, réel, double précision ou de type chaîne de caractères. Les expressions ne peuvent pas figurer dans cette liste.

Les chaînes doivent être encadrées par des guillemets si elles contiennent un séparateur (virgule, deux points, ou des espaces significatifs en début ou en fin). Dans les autres cas, les guillemets peuvent être omis.

Les diverses constantes de la liste sont séparées les unes des autres au moyen d'une virgule.

L'ensemble des constantes contenues dans les instructions **DATA** d'un programme peuvent être lues au moyen d'instructions **READ** de façon séquentielle. Le type des constantes lues doit correspondre au type des variables des instructions **READ** qui recevront les valeurs ainsi lues.

Les données d'une même instruction **DATA** peuvent être lues plusieurs fois si l'instruction **RESTORE** fait "remonter" le pointeur associé aux instructions **DATA**.

Exemple :

```
100 READ A,B,C,A$
110 PRINT A;B;C;A$
120 READ PI#
130 READ A$
140 PRINT PI#,A$

.....


200 DATA 1,2,3.14159," BELLE MARQUISE"
210 DATA 3.14159267,VIDEO

RUN
 1   2   3.14159   BELLE MARQUISE
 3.14159267   VIDEO
OK
```

Conseil :

Il est souvent préférable de ne pas disséminer les instructions **DATA** dans tout le programme mais de les rassembler par exemple en fin de programme afin d'obtenir une meilleure lisibilité.

Remarque :

Ne jamais mettre de **REM** sur une ligne de **DATA**, car elle sera considérée comme une donnée.

DEFGR$

Voir chapitre V.

DEFINT/DEFSNG/DEFDBL/DEFSTR

Syntaxe :

DEF < **type** > < **tranche d'alphabet** > [, < **tranche d'alphabet** >]...

$$< \text{type} > = \left\{ \begin{array}{l} \text{INT} \\ \text{SNG} \\ \text{DBL} \\ \text{STR} \end{array} \right\}$$

< **tranche d'alphabet** > = **lettre** [− **lettre**]

But :

Cette instruction exécutable permet de déclarer le type des variables selon la première lettre de leur nom à condition qu'il ne contienne pas de caractère de spécification de type ($, %, # et !).

Toute variable dont le nom ne contient pas un caractère de spécification de type est normalement de type réel (ou simple précision).

Cependant, si la première lettre de son nom figure dans la tranche d'alphabet, alors son type est celui indiqué par l'instruction **DEF**.

L'exécution de cette instruction peut modifier le type de variables qui auraient déjà reçu une valeur. Dans ce cas, les valeurs sont remises à zéro (ou ''à vide'' si elles deviennent le type ''chaîne'').

Exemple 1 :

```
1Ø DEFDBL L-P    'toutes les variables dont le
                  nom commence par L,M,N,O ou P
                  sont de type double précision
```

Exemple 2 :

```
1Ø DEFSTR A       'toutes les variables dont le
                   nom commence par A sont de
                   type chaîne
```

Exemple 3 :

```
1Ø DEFINT I-N
2Ø DEFDBL A-H,O-Z
```

Toutes les variables dont le nom commence par **I, J, K, L, M** ou **N** sont de type entier. Les autres sont de type double précision (sauf si elles se terminent par un caractère de spécification de type).

Exemple 4 :

```
1Ø I=1Ø
2Ø J=2Ø
3Ø DEFINT I
4Ø PRINT I;J

RUN
 Ø   2Ø
OK
```

L'exécution de l'instruction **DEFINT** a remis à zéro le contenu de la variable **I** mais pas celui de la variable **J**.

Conseil :

Pour des raisons de lisibilité, il est préférable de ne pas utiliser cette instruction pour initialiser une variable.

DEF USR et fonction USR

Syntaxe :

DEF USR [< numéro >] = < adresse mémoire >

USR [< numéro >] (< argument >)

où < **numéro** > est une expression facultative dont la valeur après arrondi éventuel à l'entier le plus proche doit être comprise entre **Ø** et **9**.

But :

L'instruction **DEFUSR** permet de définir jusqu'à 1Ø sous-programmes écrits en langage machine, que le programme **BASIC** pourra appeler au moyen de la fonction **USR**. Elle permet également le passage d'un argument du **BASIC** vers le programme assembleur et en retour du programme vers le **BASIC**.

Si le numéro est omis, la valeur Ø est prise par défaut ; < **argument** > est un paramètre, ou une expression obligatoire (réel ou fictif) qui permet le passage d'arguments entre le programme **BASIC** et le programme en langage machine.

Le sous-programme doit être écrit en respectant certaines règles.

- Cas de l'appel, le registre **A** indique le type de l'argument qui est transmis au sous-programme :

  2 entier
  3 chaîne
  4 réel
  8 réel double précision

- Le registre **X** indique l'adresse du premier octet si l'argument est de type numérique réel, et l'adresse du premier octet du descripteur si l'argument est de type chaîne (comme dans **VARPTR**). Dans le cas d'un argument entier, l'adresse du premier octet est (**X**) + 2.

- Le sous-programme doit préserver les registres suivants du **68Ø9** :

  S pointeur de pile
  DP pointeur de la page zéro.

- Il n'est pas nécessaire de préserver les autres registres.

Au retour les registres **A** et **X** doivent correspondre à l'argument retourné au **BASIC** si le programme doit effectivement retourner un argument. C'est-à-dire :

Pour A :

2	si l'argument est entier
3	si c'est une chaîne
4	si c'est un réel
8	si c'est un réel double précision

Pour **X** l'adresse suit la même convention que l'appel.

Remarque :

La partie de l'interpréteur qui effectue les calculs, utilise un "accumulateur flottant" dont les octets consécutifs sont :

FAC	pour l'exposant si le nombre est réel ou double précision
FACHO	représente les poids forts de la mantisse
FACMO	représente les poids moyens de la mantisse
FACLO	représente les poids faibles de la mantisse
DFACHO **DFACMO** **DFACML** **DFACLO**	représentent les poids faibles et la mantisse pour les nombres "doubles précisions"

En outre **FACMO** et **FACLO** sont utilisés pour les entiers.

Ainsi dans l'exécution de **USR [n]** (< **argument** >) le **BASIC** évalue l'argument et positionne les registres **A** (type) et **X** (adresse) du **68Ø9** avant d'appeler le programme en assembleur :

A = 2	**X** pointe sur **FAC**	l'argument fait 2 octets et se trouve dans **FACMO, FACLO**
A = 3	**X** pointe sur le descripteur de chaîne	le descripteur fait 3 octets
A = 4	**X** pointe sur **FAC**	l'argument fait 4 octets
A = 8	**X** pointe sur **FAC**	l'argument fait 8 octets

Lorsque le paramètre retourné par le programme assembleur est d'un type différent de celui passé par le **BASIC**, **A** doit indiquer le nouveau type et **X** doit être cohérent : si le **BASIC** avait passé un entier dans **(FACMO)** et que l'on retourne un réel, ce dernier doit être rangé dans **[FAC .... FACLO]** et **X** doit toujours pointer sur **FAC**.

Si l'on veut accéder directement à une variable **V** dans un programme assembleur, il suffit de l'appeler par :

X = USR [n] (VARPTR (V))

On trouve dans **[FACMO, FACLO]** le pointeur vers la variable (car le pointeur est un entier). Cf. **VARPTR**.

Exemple :

```
1Ø CLEAR,&H7EFF      'on se réserve l'espace
                      mémoire à partir de
                      7FØØ Hexadécimal
2Ø LOADM "ROUTINE"   'on charge un programme
                      assembleur dans cette
                      portion de mémoire
3Ø DEFUSR4=&H7FØØ    'que l'on pourra appeler
                      par USR4
.
.
.
15Ø Z=3.1415         'Z est un réel
16Ø Y=USR4(Z)        'on appelle le programme
                      et on lui passe la valeur
                      3.1415 qu'il trouvera grâce
                      aux registres A(=4) et X
                      (qui pointe sur FAC)du 68Ø9
```

Y est une variable factice si l'on n'attend pas de valeur en retour du programme assembleur ; on peut recevoir une valeur si les registres **A** et **X** sont correctement positionnés en sortie du programme assembleur.

Les données numériques rangées dans l'accumulateur flottant sont codées, selon leur type, sous la forme indiquée dans l'Annexe A (Représentation des variables en mémoire).

DELETE

Syntaxe :

DELETE [< numéro de ligne >] [– [< numéro de ligne >]]

But :

Suppression des lignes du programme.

DELETE < numéro de ligne >	supprime la ligne.
DELETE < ligne 1 > –	supprime les lignes depuis la ligne indiquée jusqu'à la dernière ligne incluse.
DELETE < ligne 1 > – < ligne 2 >	si ligne 1 < ligne 2, supprime toutes les lignes de ligne 1 à ligne 2 incluses.
DELETE – < ligne 2 >	supprime toutes les lignes depuis le début jusqu'à la ligne 2 incluse.
DELETE –	supprime tout le programme.

Exemples :

`DELETE 4Ø`	*supprime la ligne 4Ø.*
`DELETE 4Ø-1ØØ`	*supprime les lignes 4Ø à 1ØØ inclusivement*
`DELETE -4Ø`	*supprime toutes les lignes depuis le début du programme jusqu'à la ligne 4Ø comprise.*
`DELETE 4Ø-`	*supprime toutes les lignes à partir du numéro 4Ø jusqu'à la fin.*

Remarque :

Cette commande peut être insérée dans un programme : son exécution supprime des lignes du programme en cours et fait passer ensuite en mode commande.

DIM

Syntaxe :

DIM <tableau> [,<tableau>]...

avec

<tableau> = <nom de variable> [(indice [,indice]...)]

But :

L'instruction **DIM** est utilisée pour réserver de la place pour les tableaux à une ou plusieurs dimensions. Les expressions (indices) sont évaluées et les valeurs obtenues après arrondi à l'entier le plus proche indiquent les valeurs maxima que peuvent prendre les indices.

La valeur minimum que peut prendre un indice est zéro.

L'instruction **DIM** est exécutable et lors de son exécution les éléments d'un tableau numérique sont initialisés à la valeur Ø, les éléments d'un tableau de chaînes sont initialisés à "VIDE".

Si à l'exécution un indice est en dehors des limites spécifiées, l'erreur **BS** est détectée.

Il est possible d'utiliser des tableaux à 1, 2 ou 3 dimensions sans les déclarer. Dans ce cas la valeur numérique des indices ne doit pas excéder la valeur 1Ø.

Exemple 1 :

```
1Ø DIM A(15),M(12,15)
       'A:tableau à une dimension dont l'indice
        peut varier de Ø à 15. M:tableau à deux
        indices dont le premier peut varier de
        Ø à 12 et le second de Ø à 15
```

Exemple 2 : Utilisation du dimensionnement "dynamique".

```
1Ø INPUT"nombre d'éléments";N
2Ø DIM A(N)                    'réserve N+1
                                éléments pour A
```

Conseils :

Pour améliorer la lisibilité des programmes, éviter de créer des tableaux sans les dimensionner.

Ne pas dimensionner des tableaux à des valeurs supérieures à ce qui est nécessaire pour économiser de la place en mémoire.

Dans la mesure du possible, rassembler les instructions **DIM** en début de programme.

END

Syntaxe :

END

But :

Cette instruction termine l'exécution d'un programme. Elle provoque d'abord la fermeture de tous les fichiers qui étaient ouverts juste avant son exécution puis le passage en mode commande qui se traduit par l'apparition du mot **OK** sur l'écran.

L'instruction **END** ne modifie, ni le contenu des variables ni les options de **CLEAR**.

Elle peut être placée n'importe où dans le programme pour terminer l'exécution. **END** n'est pas indispensable à la dernière ligne du programme.

Exemple 1 :

```
2ØØ END
```

Exemple 2 :

```
15Ø IF ABS(X)<=EPSI THEN END ELSE GOTO 2ØØ
```

Remarque :

Contrairement à certains "**BASIC**", cette instruction est ici exécutable et peut donc figurer plusieurs fois dans un programme.

But :

ERL et **ERR** sont des variables particulières qui permettent, lorsqu'une erreur est détectée pendant l'exécution d'un programme, de connaître le numéro de la ligne de l'erreur détectée et le numéro de l'erreur afin de la traiter.

Elles sont réservées à cet usage et toute tentative d'affectation de valeur à ces variables au moyen d'une instruction d'affectation ou d'une instruction de lecture est interdite et détectée comme une erreur de syntaxe.

Lorsqu'une erreur est détectée, deux cas sont à distinguer :

aucune routine de traitement de l'erreur n'a été indiquée ; l'exécution du programme est interrompue et un message du type

?XX Error In < numéro de ligne >

est affiché à l'écran. Dans ce message, **XX** représente deux caractères correspondant aux abréviations du type de l'erreur et **< numéro de ligne >** est un nombre indiquant le numéro de la ligne où l'erreur a été détectée.

une routine de traitement de l'erreur a été mise en veille au moyen d'une instruction **ON ERROR**. Dans ce cas, il y branchement vers la routine chargée de traiter l'erreur. Le numéro de la ligne où s'est produite l'erreur est contenue dans la variable **ERL**, et **ERR** contient le numéro de l'erreur. Ainsi les instructions de la routine d'erreur peuvent connaître le type de l'erreur afin de la traiter. Par exemple, la routine pourra contenir des instructions du type :

IF ERR = 11 THEN ....

ou

IF ERL = 1ØØ THEN ...

Si l'erreur se produit dans une instruction en mode immédiat, **ERL** prend la valeur 65535.

La liste des codes d'erreurs est donnée dans l'annexe C.

Exemple :

```
1Ø ON ERROR GOTO 1ØØ     'branchement en ligne
                          1ØØ en cas d'erreur
2Ø INPUT X
3Ø A=1/X
4Ø PRINT "L'inverse est ";A
5Ø GOTO 2Ø
1ØØ REM ROUTINE D'ERREUR
11Ø IF ERR=11 THEN PRINT "Donner un nombre plus
 grand en valeur absolue"
12Ø IF ERR=6 THEN PRINT "Donner un nombre plus
petit en valeur absolue"
13Ø RESUME 2Ø            'branchement en ligne
                          2Ø après traitement
                          de l'erreur

RUN
? 2
L'inverse est .5
? Ø
Donner un nombre plus grand
en valeur absolue
?
```

Syntaxe :

ERROR < expression numérique >

But :

Cette instruction permet de simuler une erreur dont le code est soit :

un code correspondant à un code d'erreur existant (Cf. annexe C),

un code d'erreur inexistant (il n'y a pas de code d'erreur supérieur à 61)

La valeur de l'expression numérique est si nécessaire convertie en entier par arrondi pour donner le numéro de l'erreur.

Cette erreur simulée pourra soit provoquer l'édition d'un message d'erreur, soit être traitée par le programme si une instruction **ON ERROR** définit le numéro de la ligne où débute un sous-programme de traitement de l'erreur.

Si aucune instruction **ON ERROR** n'est exécutée, il y a édition d'un message d'erreur qui caractérise le type de l'erreur simulée ou, s'il s'agit d'une erreur de code inconnu de l'interpréteur BASIC, du message

?UE Error suivi du numéro de ligne

Exemple 1 :

```
10 S=10
20 T=5
30 ERROR S+T    'l'erreur de code 15 est simulée

RUN
?LS Error In 30          LS correspond au numéro
                         de code 15
```

Exemple 2 :

```
1Ø ON ERROR GOTO 1ØØØ    'branchement en 1ØØØ en
                          cas d'erreur
2Ø ERROR 6               'simulation de
                          l'erreur 6
3Ø PRINT"SUITE DE L'EXECUTION"
4Ø ON ERROR GOTO Ø       'annule l'effet de
                          l'ON ERROR précédent
5Ø END
1ØØØ REM...TRAITEMENT DE L'ERREUR 6...
1Ø1Ø IF ERR=6 THEN PRINT"ARGUMENT INCORRECT"
1Ø2Ø RESUME NEXT         'branchement à l'ins-
                          truction suivant celle
                          qui a provoqué l'erreur

RUN
ARGUMENT INCORRECT
SUITE DE L'EXECUTION
```

Remarque :

Il est préférable quand on veut définir un code d'erreur personnel de le choisir assez grand. Cela permet de garder la compatibilité du programme même si de nouveaux codes d'erreurs étaient ajoutés au système.

Syntaxe :

EXEC < adresse mémoire >

But

Cette instruction permet le branchement vers un sous-programme écrit en assembleur chargé en mémoire centrale.

Exemple :

```
EXEC 23ØØØ
```

Remarque :

Le sous-programme assembleur doit se terminer par l'instruction assembleur **RTS** pour provoquer le retour au programme BASIC et doit au préalable sauvegarder les registres S (pointeur de pile) et DP (page zéro), et restituer leur contenu avant d'exécuter **RTS**.

FOR ... NEXT

Syntaxe :

FOR < variable numérique > = X TO Y [STEP Z]

.

.

.

NEXT [<variable numérique> [, <variable numérique>]...]

X = valeur initiale ; **Y** = valeur terminale ; **Z** = pas.

But :

Ces instructions permettent d'encadrer une séquence d'instructions dont l'exécution sera répétée plusieurs fois (boucle de calcul).

La variable qui suit le mot clé **FOR** est appelée variable de contrôle de la boucle. Elle peut être de type entier ou réel mais le type double précision n'est pas autorisé (erreur **TM** détectée).

L'expression **X** après évaluation donne la valeur de départ qui sera affectée à la variable de contrôle, l'expression **Y** donne la valeur finale à ne pas dépasser et l'expression **Z** donne le pas d'incrémentation de la variable de contrôle. Si l'option "**STEP Z**" est omise, l'incrément par défaut est 1.

A toute instruction **FOR** doit correspondre une instruction **NEXT** et une seule, mais une instruction **NEXT** peut correspondre à plusieurs instructions **FOR**.

Fonctionnement des instructions FOR et NEXT :

Les expressions **X**, **Y** et **Z** sont évaluées. La valeur de **X** est affectée à la variable de contrôle puis le test suivant est effectué :

(Y − < variable de contrôle >)*Z négatif ?

Si la réponse est oui, c'est-à-dire si X > Y avec un pas positif ou X < Y avec un pas négatif, la boucle n'est pas exécutée et l'exécution se poursuit à l'instruction qui suit l'instruction **NEXT** correspondante.

Si la réponse est non, alors les instructions suivant l'instruction **FOR** sont exécutées jusqu'à la rencontre de l'instruction **NEXT** qui provoque l'incrémentation de la variable de contrôle de la valeur du pas. Puis le même test que précédemment est effectué. La boucle est exécutée jusqu'à ce que la réponse au test soit négative, ce qui provoque le branchement à l'instruction qui suit **NEXT**.

Boucles emboîtées :

Les boucles **FOR** peuvent être "emboîtées" c'est-à-dire qu'une boucle **FOR** peut contenir entièrement une autre boucle **FOR** utilisant une variable de contrôle différente mais elles ne peuvent pas être entrelacées.

Si plusieurs boucles emboîtées ont la même instruction terminale, il est possible d'utiliser la même instruction **NEXT** multiple pour toutes ces boucles. (Cf Exemple 3).

Si au cours de l'exécution, une instruction **NEXT** est rencontrée sans que l'instruction **FOR** correspondante ait été exécutée, l'erreur **NF** est détectée. Si une instruction **FOR** est utilisée sans **NEXT**, l'erreur **FN** est détectée.

Si la variable de contrôle est omise, l'instruction **NEXT** correspond à la boucle **FOR** la plus proche non déjà appariée.

Exemple 1 :

```
10 K=10
20 FOR I=1 TO K STEP 2
30 K=K+10
40 PRINT I;K
50 NEXT I

RUN
 1   20
 3   30
 5   40
 7   50
 9   60
OK
```

ce programme montre que la valeur finale de la limite de la variable de contrôle reste 10 bien que **K** soit modifiée en cours d'exécution.

Exemple 2:

```
10 I=0 : X=0
20 FOR I=3 TO 2
30 X=X+1
40 NEXT I
50 PRINT I;X

RUN
 3  0
OK
```

ce programme montre que la boucle n'est pas exécutée mais que la variable de contrôle **I** a reçu la valeur initiale 3

Exemple 3 :

```
1Ø DIM A(2Ø,2Ø)
2Ø FOR I=1 TO 2Ø
3Ø FOR J=1 TO 2Ø
4Ø A(I,J)=1
5Ø NEXT J,I
```

cette séquence initialise à 1 tous les éléments du tableau A.

Remarque :

Quant à l'intérieur d'une boucle **FOR**, il y a branchement vers un sous-programme qui peut à son tour contenir une boucle **FOR** ou faire appel à un autre sous-programme, il peut y avoir, par mégarde, des boucles qui sont en fait entrelacées car elles font appel à une même variable de contrôle. Cela conduit à des erreurs qui ne sont pas toujours faciles à localiser. Il est donc conseillé de faire attention à ce risque.

Exemple :

```
1Ø FOR I=1 TO 1Ø
...
5Ø GOSUB 5ØØ
...
9Ø NEXT
...

5ØØ FOR K= 1 TO 2Ø
...
53Ø GOSUB 6ØØ
...
56Ø NEXT K
57Ø RETURN
6ØØ FOR I=1 TO N
...
67Ø NEXT I
68Ø RETURN
```

L'exécution provoquera une erreur car la boucle **FOR** de la ligne 6ØØ est entamée avant que la boucle **FOR** de la ligne 1Ø ne soit terminée.

Pour éviter cette erreur, il faut changer le nom de l'une des variables de contrôle

GOSUB ... RETURN

Syntaxe :

GOSUB < **numéro de ligne** >
.
.
.
RETURN

But :

L'instruction **GOSUB** permet un branchement vers un sous-programme.

L'instruction **RETURN** permet de revenir au programme appelant.

L'exécution d'un sous-programme appelé par **GOSUB** débute à la première instruction de la ligne indiquée par son numéro dans l'instruction **GOSUB**. L'exécution du sous-programme se termine par l'exécution d'une instruction **RETURN**, elle provoque le retour à l'instruction qui suit immédiatement la dernière instruction **GOSUB** exécutée.

Un sous-programme peut avoir plusieurs points d'entrée. De même, il peut contenir plusieurs instructions **RETURN**.

Un sous-programme peut appeler un autre sous-programme et ainsi de suite sans autre limitation que celle provenant de la taille de la mémoire. L'appel récursif est permis ainsi que le montre l'exemple qui figure dans la remarque.

Une tentative d'exécution de l'instruction **GOSUB** vers un numéro de ligne inexistant est détectée par l'erreur **UL**.

Un sous-programme peut être placé n'importe où dans le corps du programme, mais il est souhaitable qu'il soit nettement séparé du programme principal.

Si une instruction **GOSUB** provoque le branchement vers une ligne contenant une remarque (mot clé **REM**), l'exécution se poursuit à la ligne qui suit immédiatement la ligne **REM**.

Exemple :

```
1Ø GOSUB 1ØØ
2Ø PRINT"RETOUR PREMIER APPEL"
3Ø GOSUB 1ØØ
4Ø PRINT"RETOUR SECOND APPEL"
5Ø END
1ØØ PRINT "DEBUT SOUS-PROGRAMME"
11Ø PRINT "FIN DE SOUS-PROGRAMME"
12Ø RETURN

RUN
DEBUT SOUS-PROGRAMME
FIN DE SOUS-PROGRAMME
RETOUR PREMIER APPEL
DEBUT SOUS-PROGRAMME
FIN DE SOUS-PROGRAMME
RETOUR SECOND APPEL
OK
```

Conseils :

Pour permettre une meilleure lisibilité des programmes, il est souhaitable de séparer nettement les sous-programmes du programme principal. Par exemple, les sous-programmes peuvent être rassemblés en fin de programme.

Quand un sous-programme suit immédiatement le programme principal, afin de se prémunir contre une entrée intempestive dans le sous-programme, il est souhaitable de le faire précéder par une instruction **STOP** ou **END**.

Remarque :

Il est possible de construire des sous-programmes avec appels récursifs comme le montre l'exemple suivant pour le calcul de factorielle N.

```
1Ø INPUT I
2Ø GOSUB 1ØØ
3Ø PRINT F
4Ø GOTO 1Ø
1ØØ F=1
11Ø IF I=1 THEN RETURN
12Ø F=F*I:I=I-1:GOSUB 11Ø
13Ø RETURN
```

Syntaxe :

GOTO < numéro de ligne >
ou
GO TO < numéro de ligne >

But :

L'exécution de cette instruction provoque le branchement au début de la ligne indiquée par < **numéro de ligne** >.

Si la ligne désignée contient une instruction exécutable, celle-ci sera exécutée, puis les instructions suivantes.

Si la ligne désignée ne contient pas d'instruction exécutable (par exemple une remarque), le branchement se fait sur la première instruction exécutable du numéro de ligne immédiatement supérieur au < **numéro de ligne** >.

Si une ligne contient plusieurs instructions, le branchement ne peut se faire que sur la première instruction de la ligne.

Si le numéro de ligne n'existe pas, l'erreur **UL** est détectée.

En mode immédiat, cette instruction permet de faire poursuivre au < **numéro de ligne** > indiqué l'exécution d'un programme interrompu.

Elle permet également le lancement de l'exécution d'un programme au < **numéro de ligne** > indiqué ; sous cette forme elle est donc équivalente à

RUN < numéro de ligne >

à ceci près qu'elle ne réinitialise pas les variables et qu'elle ne ferme pas les fichiers.

Exemple 1 :

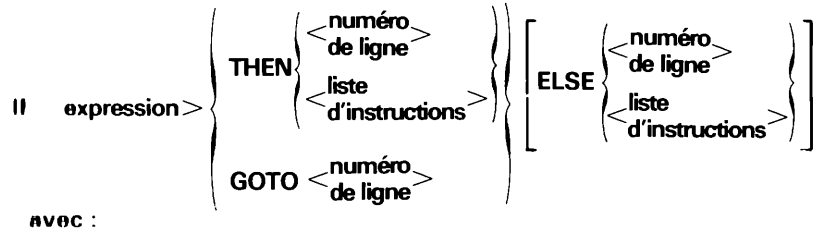
```
10 INPUT R
20 A=3.14159*R*R
30 PRINT"RAYON=";R;TAB(20);"AIRE=";A
40 GOTO 10

RUN
?5
RAYON= 5              AIRE= 78.5398
?
```

Exemple 2 :

```
GO TO 2Ø
```

branchement direct à la ligne 2Ø sans remise à zéro ou à vide du contenu des variables

Syntaxe :



avec :

liste d'instructions> = <instruction>[: <instruction>]...

But :

Cette instruction permet des branchements conditionnels selon les résultats d'un test.

L'exécution de cette instruction se fait de la façon suivante :

L'expression est d'abord évaluée.

Si le résultat n'est pas nul ou si le résultat de comparaison est "vrai", alors les instructions qui suivent **THEN** sont exécutées.

Si l'option **ELSE** est présente les instructions qui suivent **ELSE** sur la même ligne sont ignorées.

Si le résultat est nul ou si le résultat de la comparaison est "faux" il y a branchement à la ligne suivante si l'option **ELSE** est absente ou exécution des instructions de l'option **ELSE** si elle est présente.

Dans chaque cas si la liste d'instructions est remplacée par un < **numéro de ligne**>, il y a branchement à la ligne indiquée si cette ligne existe sinon l'erreur **UL** est détectée.

Si les instructions de la liste ne contiennent pas de branchements, il y a poursuite de l'exécution en séquence. Par contre, si elles contiennent une instruction de branchement, celui-ci est effectué.

Exemple :

```
1ØØ INPUT A,B,C
11Ø IF B*B-4*A*C<Ø THEN 5ØØ   'saut en 5ØØ
                               si négatif
```

Emboîtement d'instruction IF

Les instructions **IF** peuvent être emboîtées, c'est-à-dire que dans la syntaxe indiquée, il est possible de faire figurer à l'intérieur d'une instruction **IF** une ou plusieurs autres instructions **IF** à condition toutefois de ne pas dépasser la longueur d'une ligne logique (soit 255 caractères).

Exemples :

```
IF A<B THEN PRINT "A plus petit" ELSE IF A=B
THEN PRINT "égaux" ELSE PRINT "A plus grand"

IF A<=B THEN IF A=B THEN PRINT "égaux" ELSE
PRINT "A plus petit" ELSE PRINT "A plus grand"
```

L'interprétation d'instructions **IF** emboîtées est la suivante :

chaque "**ELSE**" est associé au "**THEN**" précédent le plus proche et qui n'est pas encore associé à un "**ELSE**".

Exemple :

si le nombre d'"**ELSE**" est inférieur au nombre de "**THEN**"

```
IF A<B THEN IF B<C THEN PRINT "A<C"
ELSE PRINT "B>=C"
```

Exemple :

si les nombres de "**THEN**" et d'"**ELSE**" sont égaux,

```
IF A<B THEN IF B<C THEN PRINT "A<C" ELSE PRINT
"B>=C" ELSE PRINT "A>=B"
```

Remarque :

Les tests sur des nombres réels sont parfois entachés d'une légère erreur et les tests d'égalité ne donnent pas toujours les résultats escomptés.

Exemple :

```
1Ø X=Ø
2Ø A=A+X
3Ø X=X+Ø.1              'progression de X
4Ø IF X<>1 THEN 2Ø      'test d'égalité
```

Cette séquence donne une boucle infinie car il n'y a jamais égalité. Il faut donc écrire en ligne 4Ø :

```
4Ø IF X<1 THEN 2Ø
```

Syntaxe :

INPEN X, Y

INPUTPEN X, Y

où **X** et **Y** sont des variables numériques.

X = coordonnée horizontale lue ; **Y** = coordonnée verticale lue.

But :

Ces instructions permettent de lire les coordonnées du point de l'écran visé par le crayon optique dans les conditions suivantes :

- **INPEN** effectue immédiatement la lecture
- **INPUTPEN** n'effectue la lecture que lorsque le contact du crayon optique est fermé.

La première variable reçoit la valeur de la coordonnée horizontale du point et la seconde la valeur de la coordonnée verticale. La mesure est faite en mode graphique dans les conditions suivantes :

- **Y** reçoit une valeur entière comprise entre **Ø** et **199**.
- **X** reçoit une valeur comprise entre **Ø** et **319** de façon discontinue : la mesure donne la valeur du milieu de segment de 8 points qui contient le point visé. Ainsi **X** pourra recevoir des valeurs à partir de 3 avec un pas de progression de 8 : 3, 11, 19, etc. la coordonnée horizontale est donc mesurée avec une erreur maximale de 4. Attention : sur le TO7-70, cette valeur est continue de **Ø** à **319**.

Si la mesure ne peut être effectuée, par suite :

- d'un mauvais réglage du crayon optique,
- d'une luminosité insuffisante de la zone de l'écran visée (couleur noire, par exemple).
- de la visée, par le crayon optique, d'un point situé en dehors de la zone de travail de l'écran.

la valeur − 1 est retournée pour **X** et **Y**.

Exemple :

```
1ØØ INPUTPEN X,Y
12Ø IF X<Ø OR Y<Ø THEN 1ØØ  'ou IF X<Ø THEN 1ØØ
```

Conseils :

Avant d'utiliser le crayon optique, s'assurer que la luminosité de l'écran est suffisante.

Ne pas essayer de faire lire les coordonnées de points de couleur noire ou rouge, l'élément sensible du crayon optique ne réagissant pas aux spectres de ces couleurs.

Voir également les instructions **PEN** et **ONPEN** utiles pour les programmes de mise en œuvre du crayon optique.

Syntaxe :

INPUT [;] [<chaîne de caractères>{,;}] <variable> [, <variable>]...

Le premier point virgule facultatif, suivant immédiatement **INPUT**, est sans signification. Il est autorisé afin de permettre la compatibilité avec d'autres BASIC.

But :

Cette instruction permet d'introduire des données au clavier en cours d'exécution du programme dans les conditions suivantes :

- le message éventuel "< **chaîne de caractères** >" est affiché.
  Il est suivi d'un point d'interrogation si le délimiteur est un point-virgule.
  Le point d'interrogation n'est pas affiché si le délimiteur est une virgule.
- le passage en caractères de taille normale est forcé.

L'utilisateur doit alors introduire le nombre exact de données requis par la liste des variables. Ces données doivent avoir le même type que celui de la liste sinon le système redemandera une nouvelle introduction complète de la liste des données (message **Redo**).

Les différentes données introduites sont séparées par des virgules

Les variables de la liste ne sont modifiées que si l'introduction des données est acceptée.

Si l'introduction des données est interrompue par un CNT C aucune des variables de la liste n'est modifiée. Si l'utilisateur désire poursuivre l'exécution du programme par la commande **CONT**, il devra réintroduire la liste complète des données.

L'introduction d'une chaîne peut se faire sans l'encadrer entre guillemets à condition qu'elle ne contienne ni espace en début ou en fin, ni de virgule. Si ce n'est pas le cas, il faut l'introduire sous forme encadrée entre guillemets.

Les variables de la liste peuvent être de type numérique ou chaîne de type simple ou des éléments de tableaux Elles ne peuvent pas être des tableaux.

Exemples :

```
1Ø INPUT A,B,C$
2Ø PRINT USING C$;A,B

RUN
? 1234,-5,#####  #####      ←— deux espaces entre les #
 1234     -5
OK
```

```
1Ø INPUT A$,B$,C$
2Ø PRINT A$;B$;C$

RUN
? AB ," AB D","AB,CD"
AB AB DAB,CD
OK
```

Remarque :

INPUT ne peut pas être utilisé en mode direct.

Voir chapitre IV.

INPUTWAIT

Syntaxe :

INPUTWAIT <N° de ligne> ; <durée>, [<chaîne de caractères>{;/,}] [<m>[, <n>]....]

où : **durée = < expression numérique >**

{m/n} sont des variables

But :

Cette instruction permet de lire des données, à partir du clavier, en accordant à l'utilisateur un temps limité pour introduire les données.

< **durée** > est une expression numérique dont la valeur représente le temps maximum exprimé en secondes alloué à l'utilisateur.

Le < **N° de ligne** > indique la ligne à partir de laquelle l'exécution doit se poursuivre en cas de dépassement du temps alloué.

Si le < **N° de ligne** > est omis, l'erreur **UL** est détectée.

Si un point-virgule suit la chaîne de caractères, un point d'interrogation est affiché. Si c'est la virgule, le point d'interrogation n'est pas affiché.

Exemple :

```
1ØØ INPUTWAIT 5ØØ;1Ø,A,B   'maximum 1Ø secondes
          ........
49Ø END
5ØØ PRINT "trop lent"       'utilisateur trop lent

RUN
? trop lent
OK
```

Remarque :

La forme **INPUTWAIT** < **N° de ligne** > ; < **durée** >, "", permet, d'obtenir une temporisation.

Exemple :

```
......
1ØØ INPUTWAIT 11Ø ; 5, "",        'temporisation de
                                   5 secondes
11Ø PRINT "FINI"
```

Dans ce cas, une entrée de donnée, lors de l'exécution de cette instruction, provoque une erreur.

Syntaxe :

[LET] < variable > = < expression >

But :

Cette instruction permet d'affecter une valeur à une variable.

Le mot clé **LET** est optionnel.

La variable et l'expression doivent toutes deux être soit de type chaîne soit de type numérique, sinon l'erreur **TM** est détectée.

Exemples :

```
11Ø LET D=12
12Ø E=12↑2
13Ø F=12↑4
14Ø SOMME=12+E+F
15Ø PRINT SOMME

RUN
 2Ø892
OK
```

```
1ØØ A$="BELLE"    'affectation d'une constante
11Ø B$=" "        'chaîne composée d'un espace
12Ø C$=A$+B$+"MARQUISE"
                  'affectation d'une expression
130 PRINT C$

RUN
BELLE MARQUISE
OK
```

LINE INPUT

Syntaxe :

$$\left\{\begin{matrix}\text{LINE INPUT}\\ \text{LINEINPUT}\end{matrix}\right\}[;]\ [\ <\text{chaîne de caractères}>\left\{\begin{matrix},\\ ;\end{matrix}\right\}\ ]\ <\text{variable chaîne}>$$

But :

Cette instruction permet d'introduire dans une variable chaîne, une séquence d'au plus 255 caractères, composée de lettres, chiffres, signes de ponctuation, espaces, etc.

La fin de la séquence introduite est marquée par l'appui sur la touche **"ENTREE"**. Tous les caractères frappés, caractère "retour chariot" exclu, sont affectés à la variable chaîne

Le point-virgule optionnel qui suit immédiatement le mot clé **INPUT** n'a aucune signification.

La présence de la chaîne de caractères provoque l'affichage de celle-ci avant l'introduction des caractères par l'utilisateur.

Cette instruction ne provoque pas l'apparition d'un point d'interrogation sur l'écran du téléviseur.

Exemple :

```
1Ø LINE INPUT"ENTREZ ";A$
2Ø PRINT A$

RUN
ENTREZ SANS FRAPPER
SANS FRAPPER
OK
```

Remarque :

Cette instruction est très pratique pour lire une ligne dont on ne connaît pas la structure. Il faut ensuite analyser le contenu de la variable chaîne qui reçoit le contenu de la ligne introduite.

LINE INPUT

Voir chapitre IV.

LIST

Voir chapitre IV.

LOAD

Voir chapitre IV

LOADM

Voir chapitre IV

LOCATE

Syntaxe :

LOCATE X, Y [,Z]

X = numéro de colonne ; **Y** = numéro de ligne ; **Z** = présence curseur.

Les valeurs de **X**, **Y** et **Z** après arrondi éventuel à l'entier le plus proche, doivent avoir des valeurs situées entre :

- Ø et 39 pour **X**
- Ø et 24 pour **Y**
- Ø ou 1 pour **Z**

But :

Cette instruction permet de positionner le curseur sur l'écran en mode caractère :

- **X** indique le numéro de la colonne sur laquelle le curseur doit être positionné.
- **Y** indique le numéro de la ligne sur laquelle le curseur doit être positionné.
- le paramètre **Z**, facultatif, est interprété de la façon suivante :
  - Ø le curseur ne doit pas être visible
  - 1 le curseur doit être visible (trait clignotant).

LOCATE n'est pas affectée par l'exécution d'une instruction **CONSOLE**.

Exemple :

```
1Ø LOCATE 1Ø,15                    'positionnement
                                    du curseur
2Ø PRINT"RESULTAT DES COURSES"     'affichage
```

Remarque :

Cette instruction permet en outre d'écrire à nouveau sur des lignes déjà utilisées.

Syntaxe :

MID$ (< variable chaîne >, n [,m]) = < p >

n et **m** sont des expressions numériques
p est une expression chaîne de caractères

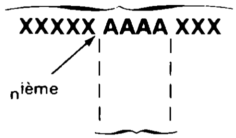
But :

Cette instruction permet de remplacer une partie des caractères d'une variable chaîne par ceux de l'expression chaîne.

Plusieurs cas peuvent être envisagés :

- Les m premiers caractères de l'expression chaîne remplacent à partir de celui de rang **n**, **m** caractères de ceux affectés à < **variable chaîne** >.

caractères de < **variable chaîne** >



m premiers caractères de
< **expression chaîne** >

Exemple 1 :

```
            6   7
1Ø  A$="MICROPROCESSEUR"
2Ø  MID$(A$,6,7)="ORDINATS"
3Ø  PRINT A$              7

RUN
MICROORDINATEUR
OK
```

Dans cet exemple, les 7 premiers caractères de "**ORDINATS**" remplacent 7 caractères de "**MICROPROCESSEUR**" à partir du 6e inclus.

- Si **m** est absent, la totalité des caractères de < **expression chaîne de caractères** > remplace à partir de celui de rang **n**, ceux affectés à la variable, sans pour autant allonger la longueur.

Exemple 2 :

```
1Ø A$="SAINT-LOUIS"
2Ø MID$(A$,7)="DENIS"
3Ø PRINT A$

RUN
SAINT-DENIS
OK
```

Dans cet exemple "**DENIS**" remplace tous les caractères de **A$** à partir du 7e inclus (**m** est absent et **n = 7**).

Dans tous les cas, **m** présent ou absent, la longueur de la suite de caractères affectée à < **variable chaîne de caractères** > n'est pas augmentée.

• < **expression chaîne** > remplaçante plus grande que la suite de caractères de < **variable chaîne** >

Dans ce cas seuls les premiers caractères remplaçants sont pris en considération.

Exemple 3 :

```
1Ø A$="TELEPATHIE"
2Ø MID$(A$,5,8)="VISIONNEUSE"
3Ø PRINT A$

RUN
TELEVISION
OK
```

Dans cet exemple, les huit premiers caractères de "**VISIONNEUSE**" doivent remplacer les caractères de "**TELEPATHIE**" à partir du 5e.

La longueur de **A$** devant rester constante seuls les six premiers caractères de "**VISIONNEUSE**" sont pris en considération.

• < **expression chaîne** > remplaçante plus courte que la suite de caractères de < **variable chaîne** >

La substitution des caractères ne s'effectue que sur une longueur égale à celle de la chaîne remplaçante.

Exemple 4 :

```
10 A$="ORGANISATIONS"
20 MID$(A$,7,6)="GRAMME"
30 PRINT A$

RUN
ORGANIGRAMMES
OK
```

Remarque :

Ne pas confondre l'instruction **MID$** avec la fonction du même nom

MOTOROFF
MOTORON

Syntaxe :

MOTOROFF

MOTORON

But :

L'instruction **MOTOROFF** arrête le défilement de la bande magnétique du lecteur-enregistreur de programmes.

L'instruction **MOTORON** permet le défilement de la bande magnétique du lecteur-enregistreur de programmes. Ce dernier doit être raccordé à l'ordinateur, sous-tension et la touche "▷"enfoncée.

Exemple :

```
MOTOROFF        ' arrêt du défilement de la
                  bande

MOTORON         ' défilement de la bande
```

Remarque :

En cas d'erreur, le moteur est arrêté automatiquement.

Voir chapitre IV.

NEW

Syntaxe :

NEW

But :

Cette commande efface le programme présent en mémoire et détruit toutes les variables. Elle ne ferme pas les fichiers et ne modifie pas les options d'une éventuelle instruction **CLEAR**.

Cette commande est utilisée pour supprimer tout programme résident avant d'entamer un nouveau travail.

Remarque :

Insérée dans un programme cette instruction de commande a pour conséquence d'annuler le programme présent en mémoire et toutes les variables.

ON ERROR GOTO

Syntaxe :

ON ERROR GOTO < **numéro de ligne** >

But :

Permet le branchement au < **numéro de ligne** > spécifié, si dans le déroulement de l'exécution d'un programme, une erreur est détectée.

Le traitement de cette erreur est alors effectué à partir du numéro de ligne indiqué.

L'instruction **ON ERROR GOTO** est donc insérée soit en début de programme, soit au début des séquences où une erreur est susceptible de se produire.

L'instruction **ON ERROR GOTO Ø** annule l'effet de **ON ERROR GOTO n**. Toute erreur ultérieure entraîne de ce fait l'apparition du message correspondant et l'arrêt de l'exécution en cours.

L'instruction **CLEAR**, entre autres choses, annule également l'effet d'une instruction **ON ERROR GOTO**.

Exemple :

```
1Ø ON ERROR GOTO 1ØØ   'branchement en ligne 1ØØ
                        en cas d'erreur
2Ø INPUT "DIVISEUR";X
3Ø PRINT 1/X
4Ø ON ERROR GOTO Ø      'annule l'effet de
                        l'ON ERROR précédent
5Ø END
1ØØ IF ERR=11 THEN PRINT"DIVISION IMPOSSIBLE"
11Ø RESUME 2Ø           'branchement en ligne 2Ø
                        après traitement de
                        l'erreur


RUN
DIVISEUR? Ø             introduction de la donnée X
DIVISION IMPOSSIBLE
DIVISEUR? 2
 .5
OK
```

Remarque :

L'exécution de tout sous-programme de traitement d'erreur doit se terminer normalement par l'instruction **RESUME**.

Conseils :

Pour tout programme dont certaines données sont susceptibles d'entraîner des erreurs, et afin d'éviter une interruption inattendue de l'exécution, une séquence de traitement des erreurs peut être prévue.

La mise au point de cette séquence pourra se faire en simulant des erreurs au moyen de l'instruction **ERROR**.

ON ... GOTO et ON ... GOSUB

Syntaxe :

ON <expression> GOTO <N° de ligne> [,<N° de ligne>]...
ON <expression> GOSUB <N° de ligne> [,<N° de ligne>]...

But :

Ces instructions permettent un branchement à une ligne dont le numéro est déterminé par la valeur de l'expression.

L'expression est évaluée et sa valeur, après conversion si nécessaire à l'entier le plus proche par arrondi, indique le rang, dans la liste, du numéro de ligne auquel le branchement devra être fait.

Par exemple, si cette valeur est 3, il y aura branchement au 3e numéro de ligne de la liste.

Pour l'instruction **ON ... GOSUB**, chaque numéro de ligne de la liste doit être un numéro d'entrée dans un sous-programme.

Si la valeur de l'expression est nulle ou supérieure au nombre de numéros de ligne de la liste, l'exécution du programme se poursuit à la ligne qui suit cette instruction et aucune erreur n'est signalée.

Si la valeur de l'expression est négative ou supérieure à 255, l'erreur FC est détectée.

Si il y a tentative de branchement à un numéro de ligne inexistant dans le programme ou si le numéro de ligne est inexistant dans la liste des numéros de ligne l'erreur **UL** est détectée.

Exemple :

```
1Ø INPUT X
2Ø ON SGN(X)+2 GOTO 1ØØ,2ØØ   'branchement selon
                                 le signe de X
3Ø PRINT"POSITIF"
4Ø GOTO 1Ø
1ØØ PRINT"NEGATIF"
11Ø GOTO 1Ø
2ØØ PRINT"NUL"
21Ø GOTO 1Ø

RUN
? -2
NEGATIF
? 1Ø
POSITIF
? Ø
NUL
?

Break In 1Ø
OK
```

Conseil :

Pour des raisons de lisibilité et de maintenance du programme, il est conseillé d'utiliser ces instructions avec une expression dont la valeur reste dans la plage normale de variation de 1 à n s'il y a n numéros de ligne.

ONPEN GOTO et ONPEN GOSUB

Syntaxe :

ONPEN GOTO < **numéro de ligne** > [, < **numéro de ligne** >]...

ONPEN GOSUB < **numéro de ligne** > [, < **numéro de ligne** >]...

But :

L'instruction **ONPEN GOTO** permet un branchement conditionnel qui dépend du numéro de la zone visée par le crayon optique.

L'instruction **ONPEN GOSUB** permet dans les mêmes conditions, un branchement conditionnel vers un sous-programme.

Des zones de l'écran ayant été définies au moyen de l'instruction **PEN**, l'exécution de l'instruction **ONPEN**... se déroule de la façon suivante :

- test de la fermeture du contact du crayon optique,
- lorsque le contact est fermé, lecture de la zone de l'écran visée par le crayon,
- branchement vers la ligne dont le numéro a le même rang dans la liste que celui de la zone définie dans l'instruction **PEN.**

Dans ces instructions, le rang dans la liste est compté à **partir de zéro.**

Si le crayon optique vise un point situé hors des zones définies, l'exécution se poursuit en séquence et aucune erreur n'est signalée.

Exemple :

```
PEN 0;(10,15)-(30,35)     'définition de zone
PEN 1;(40,15)-(60,35)
PEN 2;(70,15)-(90,35)
BOXF(10,15)-(30,35),7     'rectangle blanc
BOXF(40,15)-(60,35),7
BOXF(70,15)-(90,35),7
LOCATE 0,10               'positionne
                           le curseur
ONPEN GOTO 170,190,210    'branchement
                           conditionnel
PRINT "EN DEHORS D'UNE ZONE!"
GOTO 40
PRINT "ZONE 1:VISEE CORRECTE"
GOTO 40
PRINT "ZONE 2:VISEE CORRECTE"
GOTO 40
PRINT "ZONE 3:VISEE CORRECTE"
GOTO 40

RUN
```

Conseil :

Voir le chapitre V pour la mise en œuvre des instructions **PEN** et **ONPEN**.

OPEN

Voir chapitre IV.

PEN

Syntaxe :

PEN [< zone > [, < zone >]....]

avec

< zone > = < numéro de zone > ; [(X1, Y1) [− (X2, Y2)]]

Les numéros de zone peuvent varier de Ø à 7.

Xn, Yn = coordonnées de la ''boîte'' sélectionnée.

But :

Cette instruction permet de définir des zones rectangulaires sur l'écran et de leur attribuer un numéro, ou de supprimer des zones antérieurement définies.

L'instruction **ONPEN** permet d'effectuer un branchement selon le numéro de la zone visée par le crayon optique.

Les zones peuvent être définies en mode caractère ou en mode graphique.

- **mode caractère** : la zone est définie par la syntaxe suivante

< numéro de zone > ; (X, Y)

où **X** et **Y** correspondent aux coordonnées d'un carré en mode caractère.

Dans ce cas **X** doit être dans l'intervalle **(Ø, 39)** et **Y** dans l'intervalle **(Ø, 24)**. Les côtés du carré ainsi déterminé ont une longueur correspondant à 8 points en mode graphique (matrice 8 × 8 de représentation des caractères en taille normale).

- **mode graphique** : la zone est définie par :

< numéro de ligne > ; (X1, Y1) − (X2, Y2)

où **(X1, Y1)** et **(X2, Y2)** représentent les coordonnées en mode graphique des sommets opposés d'un rectangle (comme l'instruction **BOX**).

X1 et **X2** doivent alors être dans l'intervalle **(Ø, 319)** et **Y1** et **Y2** dans l'intervalle **(Ø, 199)**.

Une instruction **PEN** sans argument annule l'effet des instructions **PEN** précédentes : toutes les définitions de zone sont perdues.

De même, si un numéro de zone est défini sans être suivi des coordonnées, la zone est supprimée et ne peut plus être sélectionnée.

Si deux zones ont une partie commune, lorsque le crayon vise cette partie commune, les instructions **ONPEN GOTO** et **ONPEN GOSUB** effectuent le branchement vers la zone de plus petit numéro.

Exemples :

Définition de 2 zones rectangulaires :

```
10 PEN 0:(10,10)-(40,30),1:(200,10)-(250,30)
```

Définition d'une zone en mode caractère :

```
10 PEN 3:(30,15)
```

Annulation de la zone 1 définie antérieurement :

```
10 PEN 1:
```

Annulation de toutes les zones définies antérieurement :

```
10 PEN
```

Conseil :

Voir le chapitre V pour la mise en œuvre des instructions **PEN** et **ONPEN**.

PLAY

Voir chapitre VI.

POKE

Syntaxe :

POKE I, J

où **I** et **J** sont des expressions entières.

I : adresse mémoire ; **J** : valeur à écrire.

But :

Cette instruction place à l'adresse **I** un octet dont la valeur est donnée par l'expression **J**.

La valeur de l'expression **I**, après arrondi éventuel, doit se trouver dans l'intervalle **(Ø, 65535)** sinon l'erreur **FC** est détectée.

Celle de l'expression **J**, après arrondi éventuel doit se trouver dans l'intervalle **(Ø, 255)** sinon l'erreur **FC** est détectée.

Cette instruction est sans effet si on essaie de modifier un octet situé dans une mémoire morte.

Exemple :

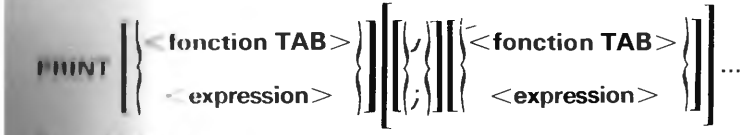
```
POKE 2ØØ,153
```

Remarques :

1 - Cette instruction peut être utilisée conjointement avec la fonction **VARPTR** pour modifier le contenu de certaines variables.

2 - La fonction **PEEK** permet de lire le contenu d'un octet dont on connaît l'adresse mémoire.

Syntaxe :



Pour réduire le temps de frappe, le mot clé **PRINT** peut être remplacé par un point d'interrogation.

But :

Cette instruction permet d'afficher, sur l'écran du téléviseur, des caractères de taille normale ou agrandie (Cf. instruction **ATTRB**).

Utilisation de caractères de taille normale :

L'exécution d'une instruction **PRINT** avec liste vide provoque le passage à la ligne suivante.

Si cette instruction comporte une liste d'expressions, celles-ci sont évaluées et éditées dans leur ordre d'apparition dans la liste.

Chaque ligne est artificiellement divisée en 3 tabulations fixes de 13 caractères chacune.

Un point-virgule provoque l'affichage de l'expression suivante juste après la valeur précédente. Une virgule provoque l'affichage de l'expression suivante juste au début de la tabulation suivante.

Si la liste d'expressions ne se termine pas par une virgule ou un point-virgule, l'affichage se termine par le passage à la ligne suivante. Si la liste d'expressions se termine par un point-virgule ou une virgule, il n'y a pas de passage à la ligne suivante.

L'affichage se fait sur la ligne courante où se trouve le curseur.

Dans le cas où une ligne de 40 caractères ne suffirait pas pour afficher les valeurs de la liste, l'affichage se fait sur plusieurs lignes consécutives. Quand une valeur ou une chaîne doit être affichée en fin de ligne, et qu'il ne reste plus assez de place, l'affichage se fait automatiquement sur la ligne suivante. Si la chaîne occupe plus de 40 caractères, elle continuera sur la ligne suivante. Dans ces cas-là, l'ordinateur considère les caractères accentués comme valant trois caractères.

Les valeurs positives sont affichées en étant précédées du caractère espace. Les valeurs négatives sont éditées en étant précédées du signe moins. Toutes les valeurs numériques sont suivies d'un espace.

Les valeurs réelles sont affichées avec au plus 6 chiffres significatifs et les valeurs "double précision" sont affichées avec au plus 16 chiffres significatifs.

L'affichage se fait sans exposant si la valeur absolue se trouve dans l'intervalle 1E − 6 à 1E- + 6 pour les réels ou 1D − 16, 1D + 16 pour les valeurs double précision. Dans les autres cas, l'affichage est fait avec exposant.

Les chaînes de caractères sont éditées en respectant la répartition des trois "tabulations" lorsqu'elles sont séparées par des virgules.

Si elles sont séparées par des points-virgules l'édition se fait sans espace entre elles. La quarantième position de l'écran peut être utilisée, mais l'édition d'une ligne complète provoque ensuite un saut de ligne.

Exemples :

```
10 A$="0123456789"
20 B$=A$+A$+A$+A$
30 PRINT B$
40 X=1:Y=2:Z=3
50 PRINT X,Y,Z
60 PRINT X;Y;Z
70 PRINT X,Y
80 PRINT X
90 PRINT X,Y;Z,
95 PRINT X

RUN
0123456789012345678901234567890123456789

 1             2             3
 1  2  3
 1             2
 1
 1             2  3          1
OK

10 PRINT"0123456789012345678901234567890123456789
20 PRINT"ABC","ABC"
30 PRINT"ABC";"ABC"
40 PRINT"ABCDEFGHIJKLMNOPQ","ABC"
```

```
#123456789#123456789#123456789#123456789#123456789

ABC                    ABC
ABCABC
ABCDEFGHIJKLMNOPQ                    ABC
```

Utilisation de caractères de double hauteur :

L'exécution de cette instruction se termine normalement par l'exécution d'un double interligne.

La première instruction **PRINT** exécutée, n'étant précédée que d'un seul interligne, risque d'''écraser'' les caractères de la ligne précédente. Il est donc préférable de la faire précéder d'une instruction **PRINT** vide pour provoquer un interligne supplémentaire.

Utilisation de caractères de double largeur :

Dans ce cas, on ne dispose que de 2Ø caractères par ligne, ce qui n'autorise plus que 2 zones d'affichage. Si la seconde zone est trop longue, l'affichage peut déborder sur la ligne suivante.

Remarques :

1 - Voir la description de la fonction **TAB** dans le chapitre III.

2 - L'utilisation de la fonction **TAB** avec des caractères double largeur est déconseillée.

3 - Les caractères accentués comptent pour trois caractères dans les chaînes.

Conseil :

Utiliser de préférence l'instruction **PRINT USING** qui permet des éditions plus élégantes.

Pour éditer une chaîne de 4Ø caractères sans provoquer d'interligne l'instruction **PRINT** doit se terminer par un point-virgule.

PRINT USING

Syntaxe :

PRINT USING <**expression chaîne**>;<**m**>[[{,;}][<**m**>]]...

avec $\begin{matrix} \mathbf{m} \\ \mathbf{n} \end{matrix}$ } **expressions**

But :

Cette instruction permet d'obtenir des éditions selon le format décrit dans l'expression chaîne.

La valeur de l'expression chaîne doit contenir des descripteurs qui décrivent le format désiré. Il existe des descripteurs pour les valeurs de type chaîne et des descripteurs pour les valeurs de type numérique.

La valeur de la chaîne utilisée pour décrire la mise en page est appelée "**IMAGE**". Les caractères de l'image non reconnus comme descripteurs seront imprimés tels quels.

Les éditions se font avec au maximum 6 chiffres significatifs exacts pour les valeurs réelles et 14 pour les valeurs "double précision"

Si l'image totale dépasse 4Ø caractères, l'édition se fait avec passage à la ligne suivante juste après le 40e caractère

Les affichages vers l'écran peuvent être faits en caractères normaux ou en caractères agrandis (double hauteur et/ou double largeur). En double largeur, les lignes de l'écran n'ont plus que 2Ø caractères.

Les paragraphes suivants présentent le rôle de chaque descripteur.

Le dernier paragraphe expose les règles de mise en œuvre de l'image.

1 - **descripteurs pour chaîne de caractères** : trois descripteurs sont disponibles pour l'édition de chaînes de caractères.

descripteur ! indique que seul le premier caractère de la chaîne doit être édité.

exemple : `PRINT USING "!"; "TOTO"`
donnera `T`

descripteur % indique la longueur de la zone d'édition qui devra être utilisée, que la chaîne à éditer soit plus longue ou plus courte. Pour cela des espaces figurent entre deux descripteurs % afin d'indiquer la longueur de la zone.

La longueur minimum de la zone est de 2 caractères et elle sera la longueur n + 2 si n espaces sont insérés entre les deux caractères

Si la zone est trop courte, elle sera éditée avec les premiers caractères de la chaîne et les caractères excédentaires ne seront pas édités.

Si la zone est trop longue, la chaîne sera éditée complètement cadrée à gauche et la zone sera complétée à droite par des caractères "espace"

exemple 1 :

```
10 A$="%%"
20 PRINT USING A$;"TOTO"
```

donnera

```
TO
```

exemple 2 :

```
10 A$="%    %"                'trois espaces entre
                               les descripteurs
20 PRINT USING A$;"TOTO"
```

donnera

```
TOTO
```

exemple 3 :

```
10 A$="%    %"                'trois espaces entre
                               les descripteurs
20 PRINT USING A$;"BELLE","MARQUISE"
```

donnera

```
BELLEMARQU
```

descripteur & indique que la zone d'édition doit avoir une longueur juste suffisante pour éditer complètement la chaîne de caractères.

exemple 1

```
PRINT USING "&";"BELLE","MARQUISE"
```

donnera

```
MARQUISE
```

exemple 2

```
PRINT USING "&";"BELLE ";"MARQUISE"
```

donnera

```
BELLE MARQUISE
```

2 - **Descripteurs pour expressions numériques** : pour l'édition de valeurs numériques, les caractères spéciaux suivants permettent de décrire les formats d'édition.

Ce caractère est utilisé pour représenter la position de chaque chiffre décimal. L'édition est faite avec justification à droite c'est-à-dire que si la zone est trop longue alors elle est complétée à gauche par des caractères ''espace''.

exemple :

```
10 A$="######"
20 PRINT USING A$;12
30 PRINT USING A$;31415
40 PRINT USING A$;-31415
```

donnera

```
    12
 31415
-31415
```

Si le nombre ne peut pas tenir dans la zone qui lui est attribuée, alors il est édité dans une zone allongée, en étant précédé du caractère %.

exemple :

```
10 PRINT USING "####";12345
```

donnera

```
%12345
```

Si la valeur numérique à éditer n'est pas entière, elle est convertie par arrondi à l'entier le plus proche avant édition.

exemple :

```
10 A$="####"
20 PRINT USING A$;12.49
30 PRINT USING A$;12.5
```

donnera

```
 12
 13
```

Un "point décimal" peut être inséré n'importe où dans une zone pour séparer la partie entière de la partie décimale.

L'édition sera faite avec toujours au moins un chiffre dans la partie entière, ce chiffre pouvant éventuellement être zéro.

Si cela est nécessaire, les valeurs numériques seront arrondies avant édition.

Comme précédemment, si la zone est trop petite pour représenter la valeur numérique de la partie entière, alors la zone est agrandie juste assez pour pouvoir représenter la valeur numérique, l'édition étant précédée du caractère %.

exemple :

```
10 A$="***.***"
20 PRINT USING A$;12
30 PRINT USING A$;-12.3454
40 PRINT USING A$;12.3455
50 PRINT USING A$;.297
60 PRINT USING A$;3145.1237
```

donnera

```
 12.000
-12.345
 12.346
  0.297
%3145.124
```

+ Dans les exemples précédents, le signe n'était édité que si la valeur à éditer était négative. Il est possible de demander explicitement que le signe + soit édité juste devant ou juste derrière la valeur numérique si celle-ci est positive.

exemple :

```
10 A$="+******.** FRANCS"
20 PRINT USING A$;12345.56
```

donnera

```
+12345.60 FRANCS
```

− Un signe − en fin de descripteur indique que les valeurs négatives doivent être éditées avec un signe − situé en fin de zone.

exemple :

```
10 A$="******.**-"
20 PRINT USING A$;-12345.67
30 PRINT USING A$; 98765.43
```

donnera

```
 45.70-
 98.43
```

** Un double astérisque, en début de descripteur, indique que les espaces en tête doivent être remplacés par des astérisques.

exemple :

```
1Ø A$="********.**"
2Ø PRINT USING A$;123.45
3Ø PRINT USING A$;12345.67
4Ø PRINT USING A$;123456.78
5Ø PRINT USING A$;1234567.89
```

donnera

```
****123.45
**12345.7Ø
*123456.78
1234567.89
```

$$ Un double dollar, en début de descripteur indique que le dernier espace non significatif avant le signe doit être remplacé par un dollar. Si la valeur à éditer est négative, le signe — est placé juste devant le caractère $, ce qui est en général peu pratique. Pour cette raison, il est possible de faire figurer dans un descripteur le double dollar en début de descripteur et le signe — à la fin. Ainsi le signe moins éventuel sera édité en fin de zone.

exemple :

```
1Ø A$="$$******.**"
2Ø PRINT USING A$;12
3Ø PRINT USING A$;123.45
4Ø PRINT USING A$;-12345.67
5Ø B$="$$******.**-"
6Ø PRINT USING B$;-12345.67
```

donnera

```
    $12.ØØ
   $123.45
-$12345.7Ø
 $12345.7Ø-
```

Il n'est pas possible d'utiliser dans un descripteur à la fois le double dollar et la notation exponentielle.

**$ Un double astérisque suivi du caractère $ permet de combiner les deux effets précédents :

— L'espace précédant le nombre est remplacé par $.

— Les autres espaces à gauche sont remplacés par *.

Les valeurs négatives ne peuvent être éditées de cette manière.

exemple :

```
1Ø A$="**$*****.**"
2Ø PRINT USING A$;1234.56

RUN
***$1234.56
OK
```

, Une virgule située juste à gauche du point décimal, indique que tous les trois chiffres de la partie entière, une virgule doit être insérée afin de séparer les puissances de 1ØØØ.

exemple :

```
1Ø A$="*********,.**"
2Ø PRINT USING A$;1234567

RUN
 1,234,57Ø.ØØ
OK
```

Une virgule en fin de descripteur est éditée comme un caractère normal.

La virgule utilisée avec la notation exponentielle est sans effet.

↑↑↑↑ Quatre flèches verticales placés en fin de descripteur indiquent que la notation exponentielle doit être utilisée.
(la flèche verticale correspond à l'accent circonflexe dans le code **ASCII**).

exemple :

```
1Ø A$="**.******↑↑↑↑"
2Ø PRINT USING A$;12345.6

RUN
 1.23456ØE+Ø4
OK
```

En cas de débordement, c'est-à-dire si la zone est trop courte pour éditer le nombre, celui-ci est édité dans une zone allongée qui commence par le caractère **%**.

Si un descripteur contient plus de 24 #, alors une erreur **FC** est détectée.

Répétition de l'image

Une image peut contenir plusieurs descripteurs et également des constantes chaînes qui seront éditées quand une instruction **PRINT USING** fera référence à l'image.

Comme une image peut contenir plusieurs descripteurs, et qu'une liste de sortie peut contenir plusieurs valeurs, plusieurs cas sont à considérer :

- l'image contient le même nombre de descripteurs qu'il y a de valeurs à éditer dans l'instruction **PRINT USING**.
- l'image contient trop de descripteurs.
- l'image ne contient pas assez de descripteurs.

Ces différents cas sont traités par les règles suivantes :

- si l'image contient juste assez de descripteurs, elle est utilisée complètement et toutes ses constantes chaînes sont éditées.
- si elle contient trop de descripteurs, seule une partie de l'image est utilisée : elle est abandonnée juste avant le premier descripteur inutile.
- si elle ne contient pas assez de descripteurs, elle est utilisée autant de fois que nécessaire à partir du début pour satisfaire la liste de sortie.

exemple :

```
10 A$="J'AI ACHETE ** POULES A **  ** F"
20 PRINT USING A$;3,25.75
30 PRINT USING A$;2
40 PRINT USING A$;3,25.75,2

RUN
J'AI ACHETE  3 POULES A 25.75 F
J'AI ACHETE  2 POULES A
J'AI ACHETE  3 POULES A 25.75 FJ'AI ACHE
TE  2 POULES A
```

Passage à la ligne suivante

Une image peut contenir jusqu'à 255 caractères et par conséquent permettre des éditions totalisant jusqu'à 255 caractères.

Pour les sorties vers l'écran dont les lignes ont 40 caractères en mode normal, un passage à la ligne est effectué, après chaque multiple de 40 caractères.

De même avec la forme "**PRINT # USING**", un passage est inséré à chaque fois qu'on atteint la fin de la ligne physique indiquée à l'interpréteur au moyen de l'instruction **OPEN** (Cf. chapitre IV).

L'exécution d'une instruction **PRINT USING** dont la liste ne se termine pas par une virgule ou un point-virgule se termine par un passage à la ligne.

Syntaxe :

READ < **variable** > [, < **variable** >]...

But :

Affecte respectivement aux variables les constantes définies dans une ou plusieurs instructions **DATA**. L'instruction **READ** ne peut être utilisée qu'en relation avec une ou plusieurs instructions **DATA**.

Les variables peuvent être de type numérique ou de type chaîne et les types des données doivent être de même nature sinon l'erreur **SN** est détectée.

Plusieurs instructions **DATA** et **READ** peuvent être insérées dans un programme. L'affectation aux variables s'effectue en commençant par la première instruction **DATA** rencontrée dans le programme.

Une même instruction **READ** peut se fournir dans plusieurs **DATA**. Inversement plusieurs instructions **READ** peuvent concerner la même instruction **DATA**. Mais dans tous les cas cités ci-dessus l'exécution d'une instruction aboutissant à une tentative de lecture de données alors que l'ensemble des constantes contenues dans les instructions **DATA** ont été lues provoque l'erreur **OD**.

L'instruction **READ** ne prend pas en compte les valeurs affectées en surnombre à une ou plusieurs instructions **DATA** mais il est possible d'utiliser plusieurs fois les données introduites par l'instruction **DATA** grâce à l'instruction **RESTORE**.

Exemple 1 :

```
1Ø FOR I= 1 TO 5
2Ø READ A(I)
3Ø PRINT A(I);
4Ø NEXT I
5Ø DATA 5,1Ø,15,2Ø,25

RUN
 5   1Ø   15   2Ø   25
OK
```

Après exécution de cette séquence, la valeur de A(1) est 5, celle de A(2) est 1Ø, etc.

Exemple 2 :

```
1Ø PRINT "NOM","PRENOM","AGE"
2Ø READ A$,B$,C
3Ø DATA "DUPOND","JACQUES",35
4Ø PRINT A$,B$,C

RUN
NOM            PRENOM         AGE
DUPOND         JACQUES         35
OK
```

Remarques :

1 - On remarquera que l'instruction **DATA** de la ligne 3Ø contient des chaînes et une valeur numérique.

2 - **READ** et **DATA** sont commodes pour incorporer des données assez volumineuses qui seraient à changer assez rarement.

Syntaxe :

$$\left\{ \begin{array}{c} \textbf{REM} \\ \textbf{,} \end{array} \right\} [< \textbf{texte} >]$$

But :

Incorporer des remarques, commentaires, titres, etc., dans un programme afin de le rendre plus compréhensible.

Cette instruction n'a aucun effet sur l'exécution du programme mais elle figure dans le listage de celui-ci.

Il est possible de se brancher par **GOTO** ou **GOSUB** sur une ligne **REM**, l'exécution s'effectuant à la première instruction (exécutable) qui la suit.

Les remarques peuvent être placées en fin de ligne en les faisant précéder d'une apostrophe qui remplace **REM**, mais une instruction **REM** écrite en toutes lettres en fin de ligne nécessite obligatoirement le séparateur :

Exemples :

```
1Ø REM...CALCUL DE LA RACINE CARREE...
2Ø Y=SQR(X)
```

ou encore

```
1Ø 'PROGRAMME DE DEMONSTRATION
2Ø Y=SQR(X) 'CALCUL DE LA RACINE CARREE
```

La remarque de la ligne 2Ø est écrite avec l'apostrophe ; écrite en toutes lettres elle nécessite le séparateur :

```
1Ø 'PROGRAMME DE DEMONSTRATION
2Ø Y=SQR(X):REM CALCUL DE LA RACINE CARREE
```

RESTORE

Syntaxe :

RESTORE [< numéro de ligne >]

But :

Relire par l'instruction **READ**, les **DATA** à partir d'un certain point. Après l'instruction **RESTORE**, le premier **READ** prend ses valeurs dans le premier **DATA** du programme si aucun numéro de ligne n'est spécifié. Dans le cas où le numéro de ligne est spécifiée **READ** prend ses valeurs dans celle-ci.

Exemple :

```
1Ø READ A,B,C
2Ø PRINT A,B,C
3Ø RESTORE              'relire les données
                         de la ligne 6Ø par
                         l'instruction de la
                         ligne 4Ø
4Ø READ D,E
5Ø PRINT D,E
6Ø DATA 1Ø,2Ø,3Ø

RUN
 1Ø            2Ø            3Ø
 1Ø            2Ø
OK
```

Après exécution de l'instruction de la ligne 4Ø, A et D prennent la valeur 1Ø, B et E la valeur 2Ø, C la valeur 3Ø.

RESUME

Syntaxe :

RESUME

RESUME NEXT

RESUME < **numéro de ligne** >

But :

Permet le retour au programme principal après traitement d'un sous-programme d'erreur (voir instruction **ON ERROR GO TO**). Cette instruction termine ce sous-programme. Insérée ailleurs que dans celui-ci, elle provoque l'apparition du message d'erreur **RE**.

RESUME L'exécution reprend à l'instruction qui a provoqué l'erreur.

RESUME NEXT L'exécution reprend à l'instruction qui suit celle qui a provoqué l'erreur.

RESUME < **numéro de ligne** >
L'exécution reprend à la ligne dont le numéro est indiqué.

Exemple :

```
1Ø ON ERROR GOTO 1ØØ   'branchement en ligne
                        1ØØ en cas d'erreur
2Ø INPUT X
3Ø PRINT SQR(X)
4Ø ON ERROR GOTO Ø      'annule l'effet de
                        l'ON ERROR précédent
5Ø END
1ØØ IF ERR=5 THEN PRINT "X NEGATIF"
11Ø RESUME 2Ø           'branchement en ligne 2Ø
                        après traitement de
                        l'erreur

RUN
? -4
X NEGATIF
```

RUN

Syntaxes :

RUN [< numéro de ligne >]
ou
RUN'' < descripteur de fichier >'' [,R]

(Cf. chapitre IV)

But :

RUN [< numéro de ligne >] permet de lancer l'exécution du programme résident.

Si le **< numéro de ligne >** est indiqué l'exécution commence à cette ligne sinon elle commence en début de programme.

RUN efface le contenu de toutes les variables de l'ancien programme avant de lancer l'exécution.

Exemple :

```
1Ø INPUT X
2Ø PRINT X

RUN
? 4
 4
OK
X = 3
OK
RUN 2Ø
 Ø
OK
```

SAVE

Voir chapitre IV.

SAVEM

Voir chapitre IV.

SCREEN

Voir chapitre V.

Syntaxe :

SCREENPRINT

But :

Recopie le contenu de l'écran sur les imprimantes graphiques références **PR9Ø-Ø4Ø** et **PR9Ø-Ø42**.

Exemple :

```
1Ø BOX(15,1ØØ)-(3Ø,2ØØ)
2Ø SCREENPRINT

RUN
```

L'ensemble, listage du programme et rectangle (résultat de l'exécution du programme) apparaissant sur l'écran est imprimé sur le papier de l'imprimante.

SKIPF

Voir chapitre IV.

STOP

Syntaxe :

STOP

But :

Suspend l'exécution du programme à l'endroit où elle se trouve et renvoie au mode commande. Cette instruction peut être utilisée en n'importe quel point du programme

Lorsqu'elle est rencontrée, le message **Break In n** apparaît mais l'exécution peut être reprise par la commande **CONT**

Au contraire de l'instruction **END**, **STOP** laisse les fichiers ouverts.

Exemple :

```
1Ø INPUT A,B
2Ø K=A↑2 : L=B↑3
3Ø STOP
4Ø M=K*L : PRINT M

RUN
? 2,3

Break In 3Ø
OK
PRINT L
 27
OK
CONT
 72
```

TRON / TROFF

Syntaxe :

TRON

TROFF

But :

TRON visualise lors de l'exécution du programme les numéros des instructions effectivement exécutées. Les numéros apparaissent entre crochets sur l'écran du terminal au fur et à mesure de l'exécution, c'est le mode "**Trace**" utilisé lors de la mise au point d'un programme.

TROFF, annule l'effet de **TRON**.

Exemple :

```
TRON
OK
10 K=10
20 FOR J= 1 TO 2
30 L= K+10
40 PRINT J;K;L
50 K=K+10
60 NEXT

RUN
 [10] [20] [30] [40]  1   10   20
 [50] [60] [30] [40]  2   20   30
 [50] [60]
OK
```

Syntaxe :

UNMASK

But :

Cette instruction permet de "démasquer" d'un seul coup toutes les zones de la fenêtre de travail qui avaient au préalable été masquées (Cf. instruction **ATTRB**).

Tous les caractères de couleur noire sur fond noir sont remis avec les attributs de couleur au moment du masquage, et l'attribut de couleur courante est positionné à cet état.

Exemple :

```
UNMASK
```

Remarques :

1 - Si la fenêtre de travail a été réduite, pour démasquer l'ensemble de l'écran, il faut donc enchaîner l'exécution de la séquence :

```
CONSOLE Ø,24
UNMASK
```

2 - En faisant varier la fenêtre de travail, on peut faire des "démasquages" sélectifs.

WAIT

Syntaxe :

WAIT <adresse>,<masque-ET> [,<masque-OU>]

But :

Cette instruction suspend temporairement l'exécution d'un programme, en attente d'un événement extérieur : action de l'opérateur ou signal envoyé par un périphérique.

Son utilisation nécessite une bonne connaissance de la structure de la mémoire du **TO7**, dont l'exposé dépasserait le cadre de ce manuel.

L'exécution est suspendue en surveillant bit par bit, à l'aide de masques, l'état d'un octet de la mémoire.

<adresse> est l'adresse de cet octet : Ø à 65535

<masque-ET> et **<masque-OU>** ont une valeur entière de Ø à 255

Le **TO7** calcule la valeur de l'expression booléenne :

<masque-ET> AND ((contenu de **<adresse>**) **XOR <masque-OU>**) et suspend l'exécution tant que cette expression est nulle.

L'exécution reprend dès que l'expression est différente de zéro.

Si **<masque-OU>** est absent, il est supposé égal à zéro, c'est-à-dire que l'expression booléenne se réduit à :

<masque-ET> AND (contenu de **<adresse>**)

Exemple :

```
1Ø PRINT "Appuyez sur le bout du crayon"
2Ø WAIT &HE7C3,2
3Ø PRINT "Merci" PRINT PLAY "DO"
4Ø PRINT "Relâchez"
5Ø WAIT &HE7C3,2,2
6Ø PRINT "Encore merci" PRINT PLAY "FA"
7Ø GOTO 1Ø
```

Dans ce double exemple, le paramètre **<masque-ET>** a pour valeur 2, c'est-à-dire le nombre binaire **ØØØØØØ1Ø**.

Sa combinaison par **AND** avec le contenu de la mémoire **&HE7C3** va isoler le second bit compté à partir de la droite, de cette mémoire. La signification de ce bit est celle de la fonction **PTRIG**.

A la ligne 2Ø, si ce bit est nul (crayon optique relâché), le **TO7** attend.

Par contre, à la ligne 5Ø l'opération **XOR** avec un même nombre binaire **ØØØØØØ1Ø** va inverser la valeur du bit. Si le crayon optique est appuyé, l'expression est nulle et le **TO7** attend qu'il soit relâché pour continuer l'exécution.

Remarque :

Pendant l'attente du WAIT, l'appui sur **CNT** **C** est inopérant.

III - FONCTIONS

INTRODUCTION

Ce chapitre présente, par ordre alphabétique l'ensemble des fonctions de la cartouche **BASIC**. D'autres fonctions sont disponibles avec la version disque.

Ces fonctions peuvent être utilisées sans déclaration préliminaire.

Les notations suivantes sont utilisées :

- Les lettres **X, Y** représentent des expressions numériques.
- Les lettres **I** et **J** représentent des expressions entières, ou si elles ne le sont pas, leur valeur est arrondie à l'entier le plus proche
- Les couples **X$** et **Y$** représentent des expressions de type chaîne de caractères.

Si le dernier caractère du nom d'une fonction est $, la fonction donne un résultat de type chaîne de caractères.

Exemples : CHR$, INPUT$

Si le dernier caractère du nom d'une fonction est une lettre, la fonction donne un résultat de type numérique.

Exemples : ABS, ASC, COS, VAL

Les fonctions mathématiques usuelles, comme **COS, EXP, LOG, RND, SIN, TAN**, etc donnent un résultat de type réel (encore appelé "simple précision").

La fonction **ABS** donne un résultat de même type que celui de l'argument.

La fonction **CDBL** donne un résultat de type double précision.

Certaines fonctions donnent un résultat de type entier comme **CINT.**

Si le type du ou des arguments de la fonction ne correspond pas à ce qui est prévu l'erreur **FC** est détectée.

ABS

Syntaxe :

ABS(X)

But :

Calcul de la valeur absolue de l'argument qui est une expression numérique de type entier, réel ou double précision.

Le type du résultat est le même que le type de l'argument.

Exemple :

```
1Ø PRINT ABS(-123.456789Ø12 );ABS(-123.45)

RUN
 123.456789Ø12   123.45
OK
```

Syntaxe :

ASC(X$)

But :

Donne, sous forme numérique, le code ASCII du premier caractère de la chaîne **X$**.

Si la chaîne **X$** est vide, l'erreur **FC** est détectée.

Si le premier caractère de la chaîne **X$** est un caractère graphique utilisateur, le code renvoyé sera 128 plus le numéro du caractère graphique.

Exemple :

```
1Ø X$="CLIENTS"
2Ø PRINT ASC(X$)

RUN
 67
OK
```

Remarque :

Ne pas oublier que les minuscules accentuées ainsi que le caractère c cédille sont représentés de façon interne par une séquence de 3 caractères. Par exemple, le caractère é est représenté par les caractères de code ASCII 22, 66 et 1Ø1.

Dans ce cas **ASC(X$)** donne seulement 22, code de la touche [ACC] . L'accès aux caractères 66 et 1Ø1 se fera par la fonction **MID$**.

CDBL

Syntaxe :

CDBL(X)

But :

Convertit la valeur de l'expression **X** en un nombre exprimé en double précision.

Si l'argument **X** n'est pas de type numérique l'erreur **FC** est détectée.

Exemple :

```
1Ø X=454.67
2Ø PRINT X;CDBL(X)

RUN
 454.67  454.67ØØ134277344
OK
```

Remarques :

1 - Les différences entre la représentation externe en mode décimal et la représentation interne en mode binaire conduit pour la partie fractionnaire à des petites erreurs de conversion.

2 - Dans l'exemple ci-dessus, la partie droite de **X** n'est par remise à Ø par **CDBL**.

Syntaxe :

CHR$(X)

But :

Donne un caractère correspondant à la valeur du code ASCII de l'argument dont la valeur doit être comprise entre Ø et 255, sinon l'erreur **FC** est détectée.

Cette fonction permet de construire des chaînes contenant des caractères de contrôle.

Exemples :

```
PRINT CHR$(7)      'correspond au signal
                    "sonnette"

PRINT CHR$(12) ;   'efface l'écran et correspond
                    à l'instruction CLS
```

Elle permet en outre de désigner les caractères spéciaux qui auraient été préalablement définis par l'instruction **DEFGR$(X)** (Cf. Chapitre V) :

CHR$(128 + X) est équivalent à **GR$(X)**

Elle permet également de construire des chaînes contenant les minuscules accentuées ou le c cédille.

Ainsi pour créer la chaîne contenant le caractère é, on peut écrire :

A$ = CHR$(22) + CHR$(66) + CHR$(1Ø1)

Exemple 1 :

```
1Ø FOR I=65 TO 68
2Ø PRINT CHR$(I);
3Ø NEXT I
```

Exemple 2 :

```
5 ATTRB 1,1              'caractères double largeur
                          et double hauteur
1Ø CLEAR,,1              'nombre de caractères
                          graphiques
15 REM...définition d'un caractère graphique...
2Ø DEFGR$(Ø)=&H81,&HØØ,&HØØ,&H18,&H18,
&HØØ,&HØØ,&H81
25 PRINT
3Ø PRINT CHR$(128)       'ou GR$(Ø)

RUN
```

OK

Remarque :

La fonction **ASC** effectue la conversion chaîne de caractères → ASCII.

Syntaxe :

CINT (X)

But :

Convertit, la valeur de l'argument **X** réel, sous forme entière. La partie fractionnaire est supprimée et le nombre est arrondi.

Si la valeur de l'argument **X** n'est pas compris dans les limites - 32768 à 32767, l'erreur **OV** est détectée (dépassement de capacité de représentation des entiers).

Exemple :

```
1Ø PRINT CINT(45.5);CINT(45.49)
2Ø PRINT CINT(-3.5);CINT(-3.49)

RUN
 46   45
-4 -3
OK
```

Voir également les fonctions **CDBL** et **CSNG** pour les autres conversions numériques et les fonctions **FIX** et **INT** qui convertissent également un argument en entier.

COS

Syntaxe :

COS (X)

But :

Calcul le cosinus de la valeur de l'argument supposé être exprimé en radians.

Exemple :

```
1Ø FOR X=Ø TO 1 STEP Ø.2
2Ø PRINT COS(X)
3Ø NEXT X

RUN
 1
 .98ØØ67
 .921Ø61
 .825336
 .6967Ø7
 .54Ø3Ø2
OK
```

Syntaxe :

CSNG(X#)

But :

Convertit la valeur de l'argument **X#** en un nombre réel "simple précision". Cette conversion se fait par arrondi et non pas par troncature

Exemple :

```
10 X#=975.341817#
20 PRINT X#;CSNG(X#)

RUN
 975.341817   975.342
OK
```

Remarque :

La fonction **CDBL** effectue la conversion simple précision → double précision.

CSRLIN

Syntaxe :

CSRLIN

But :

Cette fonction sans paramètre donne le numéro de la ligne, comptée à partir de la position **Ø**, sur laquelle se trouve le curseur.

La ligne de numéro **Ø** est la première ligne de l'écran, et la dernière ligne de l'écran a le numéro **24**.

Exemple :

```
1Ø LOCATE 5,8          'positionnement du curseur
2Ø PRINT CSRLIN

RUN

       8
OK
```

Remarque :

La fonction **POS** permet de connaître la position du curseur sur la ligne.

EOF

Voir chapitre IV

EXP

Syntaxe :

EXP(X)

But :

Cette fonction calcule l'exponentielle e^x avec e = 2.71828.
La valeur de l'argument **X** doit être inférieure ou égale à 88.Ø2969 sinon il se produit un dépassement de capacité (''overflow'') détecté par l'erreur **OV**.

Exemple :

```
1Ø FOR X=.5 TO 2 STEP .5
2Ø PRINT X,EXP(X)
3Ø NEXT

RUN
 .5            1.64872
 1             2.71828
 1.5           4.48169
 2             7.389Ø6
OK
```

FIX

Syntaxe :

FIX(X)

But :

Cette fonction convertit, par troncature, la valeur de l'argument **X** en valeur entière, mais cette valeur reste représentée sous forme réelle.

FIX(X) est donc équivalente à **SGN(X)*INT(ABS(X))**

Elle donne un résultat différent de celui de **INT** seulement lorsque l'argument est négatif.

Exemple :

```
1Ø PRINT INT(-3.6),FIX(-3.6)
2Ø PRINT INT(3.6),FIX(3.6)

RUN
-4            -3
 3             3
OK
```

Syntaxe :

FRE { **(Ø)** / **(X$)** }

But :

Cette fonction donne la place disponible, en octets, de la mémoire centrale.

FRE(Ø) donne la place disponible pour le programme et les variables.
FRE(X$) donne la place disponible pour le traitement des chaînes de caractères.

Exemple :

```
1Ø PRINT FRE(Ø),FRE(A$)
2Ø X$="12345678Ø123456789Ø"
3Ø FOR I=1 TO 8
4Ø Y$=Y$+X$
5Ø PRINT FRE(Ø),FRE(A$)
6Ø NEXT

RUN
 22596          3ØØ
 22561          28Ø
 22561          26Ø
 22561          24Ø
 22561          22Ø
 22561          2ØØ
 22561          18Ø
 22561          16Ø
 22561          14Ø
OK
```

Remarque :

L'exemple ci-dessus dépend de la taille de votre mémoire et peut ne pas donner des résultats identiques.

Ce programme provoquerait l'erreur **OS** si l'instruction de la ligne 3Ø était :

3Ø FOR I = 1 TO 9

Car il ne resterait plus assez de place pour traiter la chaîne **Y$**. Dans ce cas, il suffit de commencer le programme par l'instruction :

5 CLEAR 5ØØ

pour permettre un plus grand nombre d'itérations.

HEX$

Syntaxe :

HEX$ (X)

But :

Conversion de la valeur décimale de l'argument **X** en une chaîne de caractères représentant la même valeur en base 16. Si l'argument n'est pas entier cette fonction prend le plus grand nombre entier strictement inférieur à **X** puis fait la conversion.

Exemple :

```
10 FOR I=-50 TO 60 STEP 10
20 PRINT I;TAB(5);HEX$(I)
30 NEXT

RUN
-50   FFCE
-40   FFD8
-30   FFE2
-20   FFEC
-10   FFF6
 0    0
 10   A
 20   14
 30   1E
 40   28
 50   32
 60   3C
OK
```

Remarque :

La fonction **OCT$** permet une conversion en base 8.

Syntaxe :

INKEY$

But :

Cette fonction sans paramètre, renvoie une chaîne de longueur 1 correspondant au dernier caractère frappé au clavier (sans avoir à frapper sur la touche [ENTREE]), pendant l'exécution des instructions précédentes. Cette fonction permet d'anticiper l'envoi d'une information se limitant à un caractère.

Elle renvoie le dernier caractère frappé au clavier, ou une chaîne nulle si aucun caractère n'a été frappé. Cette fonction ne renvoie pas d'écho sur l'écran.

Tout caractère de contrôle sauf [CNT] [C] et [STOP] est transmis au programme, [CNT] [C] permettant d'interrompre l'exécution du programme et [STOP] de l'arrêter momentanément.

Exemple :

```
10 FOR I=1 TO 1000
20 NEXT
30 A$=INKEY$
40 PRINT TAB(5);A$

RUN
     W
OK
```

La touche **W** a été appuyée pendant l'exécution de la boucle **FOR**.

Remarque :

RUN n'initialise pas le registre interne du **INKEY$**. De ce fait, un caractère précédemment frappé peut être pris en compte. Dans ce cas, il convient de commencer le programme par un **C$ = INKEY$**.

INPUT$

Voir chapitre IV

INSTR

Syntaxe :

INSTR([I,] X$,Y$)

I = position dans **X$** ; **X$** = chaîne lue ; **Y$** = chaîne recherchée.

But :

Cette fonction recherche la position de la première apparition de la chaîne de caractères **Y$** dans la chaîne de caractères **X$**. Cette recherche se fait à partir du premier caractère de **X$** si le paramètre facultatif est absent. Si le paramètre **I** est présent, cette recherche se fait à partir du caractère de **X$** dont la position est égale à la valeur de l'expression **I**.

Si cette valeur excède la longueur de **X$** ou si **X$** a une longueur nulle ou encore si **Y$** ne figure pas dans **X$**, alors cette fonction donne un résultat nul.

Si **Y$** a une longueur nulle, la valeur retournée est égale à 1 ou à la valeur de **I**.

Si la chaîne **Y$** apparaît dans **X$** à la position **k**, alors la valeur retournée est **k**.

X$** et **Y$ doivent être des expressions de type chaîne.
La valeur de l'expression **I** doit être comprise entre 1 et 255 sinon l'erreur **FC** est détectée.

Exemples :

```
1Ø X$="IL FAIT BEAU, PIERRE FAIT SES DEVOIRS"
2Ø Y$="FAIT"
3Ø PRINT INSTR(X$,Y$);INSTR(6,X$,Y$);
INSTR(2,X$,"")

RUN
 4   22   2
OK
```

La première recherche donne 4 car la première séquence de ''FAIT'' apparaît en position 4.

La seconde recherche commence à partir du 6e caractère de **X$** et permet de trouver la seconde occurrence de ''FAIT''.

La troisième recherche donne la valeur du premier paramètre puisque la chaîne recherchée est vide ; en effet, pour le BASIC, **CHAINE$** est égal à '' '' + CHAINE$. De ce fait, toute chaîne commence par la chaîne vide. Ainsi, **INSTR (CHAINE$**, '' '') = 1.

INT

Syntaxe :

INT(X)

But :

Cette fonction ''partie entière'' donne le plus grand entier inférieur ou égal à la valeur de l'argument. Le résultat est un nombre réel. L'argument **X** peut être de type entier, réel ou double précision.

Exemples :

```
1Ø PRINT INT(99.89)
2Ø END

RUN
 99
OK
```

```
1Ø PRINT INT(-12.11)
2Ø END

RUN
-13
OK
```

Remarques :

Les fonctions **FIX** et **CINT** convertissent de même un argument en nombre entier mais dans des conditions différentes.

Pour obtenir l'arrondi d'un nombre à l'entier le plus proche, il suffit d'écrire :

I = INT(X + .5)

Syntaxe :

LEFT$(X$,I)

X$ = chaîne ; **I** = longueur

But :

Donne une chaîne composée des **I** premiers caractères de la chaîne **X$**. **I** ne doit pas dépasser 255. Si **I** est plus grand que la longueur de **X$**, le résultat est **X$**.

Si **I = Ø**, on obtient une chaîne vide.

Si **I < Ø**, l'erreur **FC** est détectée.

Exemple :

```
1Ø X$="INTERPRETEUR BASIC"
2Ø Y$=LEFT$(X$,12)
3Ø PRINT Y$
4Ø END

RUN
INTERPRETEUR
OK
```

Remarque :

Voir également les fonctions de sélection **MID$** et **RIGHT$**.

LEN

Syntaxe :

LEN(X$)

But :

Donne la longueur de la chaîne de caractères **X$**. Les espaces et les caractères spéciaux sont comptés.

Le paramètre peut être une constante, un nombre ou une expression chaîne.

Exemple :

```
1Ø PRINT LEN("VAL D'OISE")
2Ø END

RUN
 1Ø
OK
```

Remarque :

Les caractères de code ASCII comptent pour un caractère. Par contre, les minuscules accentuées ainsi que le c cédille comptent pour plusieurs caractères.

Exemple : **é** correspond aux caractères de code **22,66,1Ø1**
è correspond aux caractères de code **22,65,1Ø1**
â correspond aux caractères de code **22,67,97**
ï correspond aux caractères de code **22,72,1Ø5**
ç correspond aux caractères de code **22,75,99**
ü correspond aux caractères de code **22,72,117.**

Il y a donc lieu de tenir compte de ces particularités du code TELETEL dans le calcul des longueurs et dans les comparaisons.

Syntaxe :

LOG (X)

But :

Calcul du logarithme népérien de l'argument **X**. **X** doit être positif sinon l'erreur **FC** est détectée.

Exemple :

```
]% PRINT LOG(5)
]% END

 1.60944
]%
```

MID$

Syntaxe :

MID$(X$,I[,J])

X$ = chaîne ; **I** = position ; **J** = longueur

But :

Extraire une sous-chaîne de longueur **J** de la chaîne de caractères commençant au caractère de rang **I**.

I doit être compris entre 1 et 255 et **J** peut être compris entre Ø et 255. Si **J** est absent ou si **J** est trop grand, la sous-chaîne va jusqu'à la fin de **X$** ; si **I>LEN(X$)** la fonction donne une chaîne vide. Si **I = Ø**, le message d'erreur **FC** est détecté.

Exemple :

```
1Ø A$="ECHEC"
2Ø B$="PAT ET MATH"
3Ø PRINT A$;MID$(B$,4,7)
4Ø END

RUN
ECHEC ET MAT
OK
```

Utilisation du **MID$** pour transformer une valeur numérique en chaîne sans le blanc de tête :

```
10  A = 12345
20  A$ = MID$ (STR$(A),2)
30  PRINT "A = " ; A  "A$ = "  A$

RUN
A = 12345      A$ = 12345
```

Voir également les fonctions **LEFT$** et **RIGHT$**

Remarque :

Ne pas confondre la fonction **MID$** avec l'instruction du même nom.

Syntaxe :

OCT$(X)

But :

Cette fonction convertit la valeur décimale de l'argument en une chaîne représentant la même valeur exprimée en base 8.

La valeur de l'argument est convertie si nécessaire en entier avant conversion.

Exemple :

```
10 FOR I=-40 TO 40 STEP 10
20 PRINT I;TAB(5);OCT$(I)
30 NEXT

RUN
-40   177730
-30   177742
-20   177754
-10   177766
 0    0
 10   12
 20   24
 30   36
 40   50
OK
```

Remarque :

La fonction **HEX$** convertit la valeur de l'argument en une chaîne représentant la même valeur exprimée en base 16.

PEEK

Syntaxe :

PEEK(I)

But :

Donne le contenu de l'adresse mémoire spécifiée par **I**.

I est compris entre Ø et 65 535 inclus

Le résultat est un entier compris entre Ø et 255.
Cette fonction est l'opération inverse de l'instruction **POKE**.

Exemple :

```
PRINT PEEK(1Ø)
 82
OK
```

Voir chapitre V.

POS

Syntaxe :

POS[(<numéro de canal>)]

But :

Donne la position du curseur dans la ligne au moment de son exécution.
La position la plus à gauche est numérotée Ø.

< **numéro de canal** > indique le périphérique concerné (Cf Ch IV).
Si < **numéro de canal** > = Ø ou est omis, la fonction renvoie la position du curseur sur l'écran du terminal. Dans ce cas les valeurs limites sont Ø et 39.

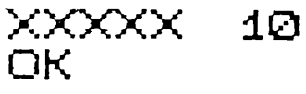
En mode caractère double largeur ou double taille (Cf. **ATTRB**) ces limites incluent un nombre moitié de caractères.
Si le canal n'est pas ouvert, l'erreur **NO** est détectée.

Exemple : 1

```
10 FOR I=1 TO 1Ø
20 PRINT "X";
30 NEXT
40 PRINT POS(Ø)  'affiche la position du cur-
                 seur au moment de l'exécution
                 de cette instruction

RUN
XXXXXXXXXX 1Ø
Ok
```

Exemple : 2

```
5 ATTRB 1,1     'caractères double hauteur et
                 double largeur
1Ø PRINT
2Ø PRINT"XXXXX";:A=POS(Ø)
3Ø PRINT A

RUN
```



```
OK
```

Pour une position de curseur identique à celle de l'exemple ci-dessus cinq caractères seulement sont affichés.

Syntaxe :

PTRIG

But :

Cette fonction, sans argument, donne la valeur "**VRAI**" notée **– 1** si le contact du crayon optique est fermé et la valeur "**FAUX**" notée **Ø** dans le cas contraire.

Exemple :

```
5 CLS                'efface l'écran
1Ø IF PTRIG GOTO 4Ø
2Ø LOCATE Ø,1Ø       'positionnement du curseur
25 PRINT"APPUYER LE CRAYON"
3Ø GOTO 1Ø
4Ø COLOR 1           'couleur rouge
5Ø LOCATE Ø,1Ø
6Ø PRINT"OK:CONTACT ETABLI"
7Ø GOTO 1Ø
```

RIGHT$

Syntaxe :

RIGHT$ (X$,I)

X$ = chaîne ; **I** = longueur

But :

Donne une sous-chaîne composée des **I** derniers caractères de la chaîne **X$**. **I** ne doit pas dépasser 255. Si **I** est plus grand que la longueur de **X$**, toute la chaîne est prise. Si **I = Ø** on obtient une chaîne vide.

Si **I<Ø**, l'erreur **FC** est détectée.

Exemple :

Syntaxe :

RND[(X)]

But :

Cette fonction donne un nombre réel, pseudo-aléatoire uniformément réparti dans l'intervalle (Ø, 1), Ø et 1 exclus.

Si l'argument **X** est omis ou s'il est positif et ce quelle que soit sa valeur, les appels successifs à cette fonction au cours d'une même exécution d'un programme donnent des nombres aléatoires différents :
Une nouvelle exécution du programme, donne la même séquence de nombres (voir exemple ci-dessous).

Si l'argument **X** est nul ou négatif, les appels successifs à cette fonction au cours d'une même exécution de programme, donnent une valeur identique. Celle-ci dépend de l'argument.

Exemple :

Cinq appels successifs à la fonction **Y = RND** donnent une série de cinq valeurs différentes.
Une nouvelle exécution de ce programme produirait la même séquence.

Conseil :

Des séquences de nombres aléatoires différentes à chaque exécution peuvent être obtenues en introduisant un facteur aléatoire, tel le temps de réponse à une question.

Exemples :

```
1Ø PRINT "APPUYER SUR UNE TOUCHE"
2Ø X = RND
3Ø IF INKEY$ ="" GOTO 2Ø
4Ø PRINT RND;RND;RND

RUN
APPUYER SUR UNE TOUCHE
 .594972   .697368   .47432
OK
```

Ce programme exécute n fois la fonction **RND**.

```
RUN
APPUYER SUR UNE TOUCHE
  .383799   .664351   .43Ø9Ø3
OK
```

Syntaxe :

SCREEN(X,Y)

X = Abscisse caractère ; **Y** = ordonnée caractère

But :

Cette fonction donne le code **ASCII** du caractère affiché sur la ligne de numéro **Y** et en position **X**.

X et **Y** sont des expressions numériques dont les valeurs, après arrondi éventuel à l'entier le plus proche, doivent se trouver dans la plage (Ø,39) pour **X** et (Ø,24) pour **Y**.

Si aucun caractère reconnaissable n'est affiché à l'endroit indiqué, cette fonction donne la valeur zéro. Seuls les caractères standards sont reconnus.

Pour les minuscules accentuées et le c cédille, les codes suivants sont attribués :

128	ç	134	ê	14Ø	ö
129	á	135	ë	141	û
13Ø	â	136	è	142	ü
131	ä	137	î	143	ù
132	à	138	ï		
133	é	139	ô		

Exemple :

```
1Ø CLEAR,,2
2Ø DEFGR$(1)=128,64,32,16,8,2Ø,66,129
                        'définit un caractère
3Ø CLS
4Ø LOCATE 2,4           'positionne le curseur
5Ø PRINT "A";GR$(1)     'affichage du caractère
                         défini
6Ø LOCATE Ø,6
7Ø PRINT SCREEN(2,4);SCREEN(3,4)

RUN

 A

 65  Ø
OK
```

La lettre A est reconnue, mais pas le caractère GR$(1).

Remarques :

1. Ne pas confondre la fonction **SCREEN** avec l'instruction du même nom.
2. Les zones de l'écran écrites en caractères double hauteur ou double largeur ou encore au moyen de caractères définis par l'utilisateur ne sont pas reconnues.
3. La présence du curseur sous un caractère analysé par **SCREEN** peut provoquer une erreur.
4. Il est recommandé d'éteindre le curseur avant le **SCREEN**.

Syntaxe :

SGN(X)

But :

Donne le signe de **X**.

argument **X>Ø**, **SGN(X) = 1**
argument **X = Ø**, **SGN(X) = Ø**
argument **X<Ø**, **SGN(X) = − 1**

Exemple :

```
LØ PRINT SGN(5),SGN(Ø),SGN(-1)

RUN
 1             Ø             -1
OK
```

Remarque :

Cette fonction peut être très utile dans certains cas. Par exemple, dans le cas de la résolution de l'équation du second degré on pourra écrire

```
1ØØ DELTA=B*B-4*A*C
11Ø ON SGN(DELTA)+2 GOTO 12Ø,2ØØ,3ØØ
12Ø PRINT"RACINES COMPLEXES"
                    .
                    .
                    .
```

SIN

Syntaxe :

SIN(X)

But :

Calcul du sinus dont l'argument est exprimé en radians. Le résultat est donné en simple précision.

Exemple :

```
1Ø PRINT SIN(1.5)
2Ø END

RUN
 .997495
OK
```

Syntaxe :

SPC(X)

But :

Création de **X** blancs. **SPC** ne peut être utilisé que dans une instruction **PRINT**. L'argument **X** est compris entre Ø et 255.

Exemple :

```
]Ø PRINT"A"SPC(15)"B"
]Ø END

]UN
.                B
]K
```

Remarque :

SPC affiche un nombre **X** d'espaces à partir de la position du curseur, tandis que **TAB** affiche des espaces jusqu'à une position définie.

Il ne doit pas y avoir d'espace entre **SPC** et **(**, sinon **SPC** est pris comme un nom de tableau.

SQR

Syntaxe :

SQR(X)

But :

Calcul de la racine carrée de l'argument. L'argument est positif ou nul et le résultat est donné en simple précision.

Si l'argument est négatif l'erreur **FC** est détectée

Exemple :

```
10 FOR X= 10 TO 25 STEP 5
20 PRINT X,SQR(X)
30 NEXT X
40 END

RUN
 10            3.16228
 15            3.87298
 20            4.47214
 25            5
OK
```

Syntaxe :

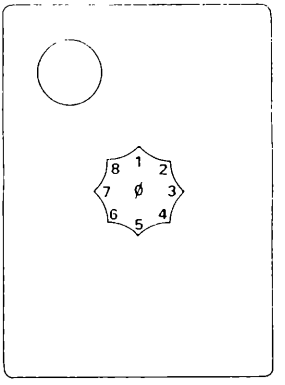
STICK(I)

où **I** peut prendre les valeurs Ø ou 1.

But :

Cette fonction donne une valeur entière de Ø à 8 selon la position de la manette de jeu de numéro **I**. Ces valeurs sont données dans le tableau suivant :

position	neutre	Ø
position	9Ø degrés	1
position	45 degrés	2
position	Ø degrés	3
position	– 45 degrés	4
position	– 9Ø degrés	5
position	– 135 degrés	6
position	18Ø degrés	7
position	135 degrés	8



Exemple :

```
.....
1ØØ A=STICK(1)
11Ø ON A GOTO ...
```

Remarque :

Voir **STRIG**

STRIG

Syntaxe :

STRIG(I)

Où **I** ne peut prendre que la valeur Ø ou 1.

But :

Cette fonction donne l'état du bouton de la manette de jeu.
I représente le numéro de la manette

I = Ø manette de numéro Ø
I = 1 manette de numéro 1

Elle renvoie le résultat :

Ø si le contact est ouvert
– 1 si le contact est fermé (bouton appuyé).

L'erreur **FC** est détectée si la valeur de **I** est négative ou supérieure à 1 (après arrondi).

Remarque :

Voir **STICK**

Syntaxe :

STR$(X)

But :

Convertit la valeur de l'expression en une chaîne de caractères composée de chiffres du code **ASCII** et représentant la même valeur que celle de l'expression.

Exemple :

```
10 A=68.5 : B=-71
20 PRINT STR$(A)
30 PRINT STR$(B)
40 PRINT STR$(B)+STR$(A)

RUN
 68.5
-71
-71 68.5
OK
```

Voir également la fonction **VAL**.

TAB

Syntaxe :

TAB (I)

But :

Cette fonction qui ne peut être utilisée que dans une instruction **PRINT** ou **PRINT #** positionne le curseur de l'écran ou de l'imprimante.

Le positionnement s'effectue sur la colonne dont le numéro est donné par l'expression **I**, après arrondi à l'entier le plus proche si nécessaire, tout en respectant les règles de tabulation définies par les séparateurs ; et , (Cf. instruction **PRINT**).

- Si la position à atteindre est à gauche de la position courante, un saut de ligne est effectué au préalable.
- Dans le cas d'un affichage sur l'écran, le nombre de positions est limité à 40 ; si la valeur de l'expression **I** est supérieure à 40, le reste de la division de **I** par 40 donne la prochaine position du curseur (positionnement modulo 40).
- Dans le cas d'une sortie sur imprimante, le nombre maximum de positions (largeur) peut être modifié lors de l'ouverture du canal imprimante (Cf. instruction **OPEN**). Si **I** est supérieur à cette largeur, le reste de la division de **I** par la largeur donne la position du curseur.
- Si la valeur de **I** est négative tout se passe comme si elle était nulle.

En mode double largeur, le positionnement est effectué en tenant compte du fait que les caractères ont une largeur double

Exemple :

```
10 A$="0123456789012345678901234567890123456789"
20 PRINT A$
30 PRINT TAB(5);"A",TAB(10);1
40 PRINT TAB(5),2
50 PRINT TAB(5);3

RUN
01234567890123456789012345678901234567890

     A
          1
              2
       3
OK
```

Ligne 30 : Le caractère **A** est affiché en colonne 5. La virgule après le **A** fait progresser le curseur dans la zone de tabulation suivante c'est-à-dire en colonne 13 ; de ce fait l'affichage du **1** en colonne **1Ø** ne peut se faire qu'à la ligne suivante ; l'édition du **1** est précédée d'un espace en colonne **1Ø**.

Ligne 40 : **TAB (5)** positionne le curseur en colonne 5 puis la virgule le fait progresser en colonne 13 où le **2** est édité précédé d'un espace.

Ligne 50 : Le **3** est édité en colonne 5, précédé d'un espace le point-virgule ne déplaçant pas le curseur.

Remarque :

Contrairement à certains **BASIC**, la première colonne a pour rang **Ø** au lieu de **1**. Cela permet d'être compatible avec les instructions graphiques comme **BOX, BOXF, LINE** et **PSET**.

Il ne doit pas y avoir d'espace entre **TAB** et **(**, sinon **TAB** est pris comme un nom de tableau. **TAB** affiche des espaces jusqu'à une position déterminée tandis que **SPC** affiche un certain nombre d'espaces.

TAN

Syntaxe :

TAN(X)

But :

Calcul de la tangente dont l'argument est exprimé en radians. Le résultat est donné en simple précision. Si la valeur du résultat dépasse ''l'infini machine'' 1E38 l'erreur **OV** est détectée.

Exemple :

```
1Ø PRINT TAN(1.5)
2Ø END

RUN
 14.1Ø14
OK
```

VAL

Syntaxe :

VAL (X$)

But :

Donne la valeur numérique de la chaîne de caractères **X$** à condition que **X$** représente un nombre sous forme de caractères ASCII. Si le premier caractère n'est pas un caractère décimal, un signe +, un signe −, un espace ou un point le résultat est égal à zéro.

Exemples :

```
10 X$="15 RUE"
20 A=VAL(X$)
30 PRINT A
40 END

RUN
 15
OK
```

```
10 PRINT VAL("15");VAL("18");VAL("AB")
20 END

RUN
 15   18   0
OK
```

VARPTR

Syntaxe :

VARPTR (<nom de variable>)

But :

Cette fonction donne l'adresse du premier octet de la variable indiquée. L'adresse de la variable doit avoir été préalablement déterminée par une instruction d'affectation, une lecture ou par une instruction **DIM** pour les tableaux sinon l'erreur **FC** est détectée.

L'adresse est donnée en représentation décimale sous forme d'un entier compris entre − 32768 et 32767. Si l'adresse fournie est négative, ajouter + 65536 pour obtenir l'adresse réelle.

Exemple :

```
10 DIM B(50,31),A(20)
20 FOR I=1 TO 20
30 X=VARPTR(A(I))
40 IF X<0 THEN X=X+65536
50 PRINT I;TAB(5);X
60 NEXT I

RUN
 1      32785
 2      32789
 3      32793
 4      32797
 5      32801
 6      32805
 7      32809
 8      32813
 9      32817
 10     32821
 11     32825
 12     32829
 13     32833
 14     32837
 15     32841
 16     32845
 17     32849
 18     32853
 19     32857
 20     32861
OK
```

Ce programme permet de vérifier que les adresses des réels progressent par pas de 4.

Remarque :

Cette fonction peut être utilisée pour obtenir l'adresse d'une variable qui sera transmise à un sous-programme écrit en langage 68Ø9.

Voir également, pour la représentation des variables en mémoire l'annexe A.

IV - ENTREES/SORTIES ET FICHIERS

IV.1 GENERALITES

Le BASIC du système **T07** permet la gestion d'entrées/sorties généralisées assurant les transferts d'informations entre l'ordinateur et les périphériques.

Ces transferts concernent :

- des fichiers programmes
- des fichiers de données
- des fichiers binaires

et sont effectués sous le contrôle d'instructions d'entrées/sorties décrites dans ce chapitre.

Ces instructions permettent de réaliser tous les échanges possibles entre plusieurs sources d'information et l'ordinateur. En particulier, le même jeu d'informations pourra être dirigé indifféremment vers un périphérique ou un autre par l'utilisation des descripteurs de fichiers (cf § IV.2).

Le tableau de la page suivante présente l'ensemble des instructions, fonctions ou commandes relatives aux opérations d'entrées/sorties généralisées.

IV.2 DESCRIPTEUR DE FICHIER NOM DE FICHIER

Les renseignements fournis à l'ordinateur quant au choix du périphérique utilisé et du fichier manipulé sont contenus dans une chaîne appelée DESCRIPTEUR DE FICHIER ou sous une forme plus réduite appelée NOM DE FICHIER.

IV.2.1 DESCRIPTEUR DE FICHIER

La syntaxe générale du descripteur de ficher est

[<type de périphérique> :] [(<option>)] [<nom de fichier>]

- Le type de périphérique dépend du périphérique à utiliser :

KYBD	pour l'utilisation d'un clavier.
SCRN	pour l'utilisation de l'écran.
CASS	pour l'utilisation du lecteur-enregistreur de programmes.
LPRT	pour l'utilisation d'une imprimante à interface parallèle.
COMM	pour l'utilisation de la voie série.

Le caractère **:** est obligatoire après le <**type de périphérique**> si celui-ci est explicite.

Lorsque le <**type de périphérique**> est omis, sont pris par défaut :

l'écran	avec l'instruction **LIST** sans nom de fichier
le lecteur-enregistreur de programmes sauf si un lecteur de disquettes est connecté	avec les instructions **SAVE, LOAD, RUN, OPEN, MERGE, LIST** avec fichier

INSTRUCTIONS FONCTIONS OU COMMANDES	PERIPHERIQUES				
	CLAVIER	ECRAN	LECTEUR ENREGISTREUR	IMPRIMANTE PARALLELE	VOIE SERIE
CLOSE	●	●	●	●	●
EOF			●		
INPUT #	●		●		●
INPUT $	●		●		●
LINE INPUT #	●		●		●
LIST		●	●	●	●
LOAD			●		
LOADM			●		
MERGE			●		
OPEN	●	●	●	●	●
PRINT #		●	●	●	●
PRINT # USING		●	●	●	●
RUN			●		
SAVE		●	●	●	●
SAVEM			●		
SKIPF			●		

Le terme **[(<option>)]** dépend du périphérique utilisé : imprimante à interface parallèle ou utilisation de la voie série. Il est inutile avec les autres périphériques.

Imprimante à interface parallèle

Dans ce cas l'option représente le nombre de caractères par ligne : entre 1 et 255. Si elle est omise le nombre de caractères par ligne est égal à 4Ø.

Voie série

L'option est un nombre entier indiquant la vitesse de transmission, le nombre de bits par caractère et le nombre de caractères par ligne selon la convention suivante :

1er chiffre Vitesse de transmission	2e chiffre nombre de bits transmis pour un caractère	3e chiffre et suivants nombre de caractères par ligne
1 pour 11Ø bauds **2** 3ØØ **3** 6ØØ **4** 12ØØ **5** 24ØØ **6** 48ØØ	**7** → transmission sur 7 bits **8** → transmission sur 8 bits	compris entre Ø et 255

Exemple : LIST"COMM:(588Ø)" transmet un programme vers la voie série à 2 4ØØ bauds avec des caractères de 8 bits et des lignes de 8Ø caractères.

Si l'option est absente les valeurs prises par défaut sont :
transmission à 2 4ØØ bauds
caractère de 7 bits
4Ø caractères par ligne.

– Le <**nom de fichier**> est expliqué ci-dessous

IV.2.2. NOM DE FICHIER

Le nom de fichier est composé du nom lui-même suivi de façon optionnelle par un suffixe :
le nom comporte un à huit caractères à l'exclusion des virgules, du zéro binaire, du caractère 255 et du point.
le suffixe optionnel comporte un point suivi de un à trois caractères répondant aux mêmes réserves que le nom.

Exemple de nom de fichier :
CRAPS. JEU
PYTHAG. MAT
ESSAI

En l'absence de suffixe donné par l'utilisateur, l'interpréteur BASIC ajoute un suffixe qui est :
DAT pour les fichiers de données.
BAS pour les fichiers-programmes.
BIN pour les fichiers binaires.

IV.3 FICHIERS-PROGRAMMES

Les fichiers-programmes contiennent des programmes BASIC, en général stockés sur cassette magnétique.

Trois formes de stockage de programmes pour la création de fichiers sont possibles :

- Forme habituelle condensée (binaire)
- Forme « ASCII » non condensée.
  La lecture d'un programme stocké sous cette forme est similaire à celle de tout fichier séquentiel ; cette forme permet en outre sa fusion avec un programme BASIC présent en mémoire, au moyen de la commande **MERGE.**
- Forme protégée contre le listage (forme codée évitant tout décryptage).
  Sous cette forme le programme peut à nouveau être chargé et exécuté mais ne peut plus être ni listé ni modifié. Les instructions AUTO, POKE ou la fonction PEEK ne sont pas autorisées.

Plusieurs méthodes permettent le stockage, l'édition ou le lancement de l'exécution des programmes. Les instructions et commandes données ci-après décrivent ces méthodes

Remarque :

Le descripteur de fichier peut être contenu dans un nom de variable (pour **RUN**, le faire précéder de `" "` +)

Syntaxe :

LIST ["<descripteur de fichier>",] [<plage de numéros de ligne>]

But

L'instruction **LIST** permet d'éditer sur le ou les périphériques une liste du programme présent en mémoire.

Les périphériques peuvent notamment être :

- l'écran
- Lecteur-enregistreur de programmes
- Imprimante à interface parallèle
- Voie série

Utilisation de l'écran

L'écran est le périphérique pris par défaut lorsque le <**descripteur de fichier**> est omis.

Syntaxe :

LIST [<numéro de ligne>][– [<numéro de ligne>]]

But :

Cette instruction provoque l'affichage en totalité ou en partie du programme présent en mémoire.

LIST	visualise tout le programme.
LIST <numéro de ligne>	affiche seulement la ligne indiquée.
LIST <numéro de ligne> –	affiche les lignes du programme depuis la ligne indiquée jusqu'à la fin.
LIST – <numéro de ligne>	affiche les lignes du programme depuis le début jusqu'à la ligne indiquée.
LIST <numéro de ligne> – <numéro de ligne>	affiche les lignes du programme dont les numéros sont dans l'intervalle indiqué.

Exemples :

`LIST 15Ø` - *affiche de la ligne 15Ø incluse à la fin du programme.*

`LIST -1ØØØ` *affiche le début du programme jusqu'à la ligne 1ØØØ.*

`LIST 15Ø-1ØØØ` *affiche une partie du programme comprise entre les lignes 15Ø et 1ØØØ incluses.*

Utilisation du lecteur-enregistreur de programmes

Syntaxe :

LIST "CASS :<nom de fichier>"[,<plage de numéros de ligne>]

But :

Permet l'enregistrement sur bande magnétique, sous forme codée ASCII d'une partie ou de la totalité du programme présent en mémoire.

Si le terme <**plage de numéros de ligne**> est omis la totalité du programme est enregistrée.

Exemples :

`LIST "CASS:ANNEE"` *enregistre, sous le nom "**ANNEE**" et sous forme codée ASCII la totalité du programme présent en mémoire.*

`LIST "CASS:ANNEE",3Ø-2ØØ` *enregistre, sous le nom "**ANNEE**" et sous forme codée ASCII toutes les lignes du programme comprises entre la ligne 3Ø et la ligne 2ØØ incluses.*

Utilisation d'une imprimante à interface parallèle.

Syntaxe :

LIST "LPRT :[(<m>)]" [,<p>]

avec **m** nombre de caractères par ligne.
p plage de numéros de ligne.

But :

Permet l'édition d'une partie ou de la totalité d'un programme présent en mémoire, sur une imprimante.

Si le terme du <**nombre de caractères par ligne**> est omis l'édition du programme s'effectue avec 4Ø caractères par ligne.
Si le terme <**plage de numéros de ligne**> est omis la totalité du programme est éditée sur l'imprimante.

Exemples :

Les imprimantes de références **PR9Ø-Ø4Ø** et **PR9Ø-Ø42** permettent une édition avec des lignes de 4Ø caractères maximum.

Pour éditer, sur une imprimante, la totalité du programme présent en mémoire à raison de 4Ø caractères par ligne il suffit de frapper la commande :

```
LIST "LPRT:"
```

L'utilisation de l'imprimante à impact de référence **PR9Ø-Ø8Ø** permet une édition avec des lignes de 8Ø caractères maximum.

```
LIST "LPRT:(8Ø)"
```

édite la totalité du programme avec des lignes de 8Ø caractères.

```
LIST "LPRT:(6Ø)",3Ø-2ØØ
```

édite avec 6Ø caractères par ligne, l'ensemble des lignes du programme dont les numéros vont de 3Ø à 2ØØ inclus

Utilisation d'une voie série

Syntaxe :

LIST "COMM :[(<option>)][<nom de fichier>]" [,<m>]

avec **m** plage de numéros de ligne

But :

Permet d'envoyer tout ou partie d'un programme vers la voie série.
L'option a été décrite au paragraphe IV.2.

Exemple :

```
LIST "COMM:(588Ø)"
```

Transmet un programme vers la voie série à 24ØØ bauds avec des caractères de 8 bits et des lignes de 8Ø caractères.

```
LIST "COMM:(578Ø)",1ØØ-15Ø
```

Transmet vers la voie série à 24ØØ bauds sur 7 bits et avec 8Ø caractères par ligne, l'ensemble des lignes comprises entre celle de numéro 1ØØ et celle de numéros 15Ø incluses.

LOAD

Syntaxe :

LOAD ["<descripteur de fichier>"[,R]]

<**descripteur de fichier**> est une constante chaîne entre guillemets, une variable chaîne ou une expression chaîne.

But :

Charger en mémoire un programme.

Le <**descripteur de fichier**> se réduit à <**nom de fichier**> avec l'utilisation du lecteur-enregistreur de programmes, celui-ci étant alors le périphérique pris par défaut.

Le <**nom de fichier**> doit être identique à celui sous lequel le programme a été sauvegardé au moyen des instructions **SAVE** ou **LIST**.

La commande **LOAD** ferme tous les fichiers ouverts, détruit le programme qui était éventuellement présent en mémoire puis effectue le chargement proprement dit.

Toutefois, avec l'option **R**, les fichiers éventuellement ouverts ne sont pas fermés, le programme est chargé et l'exécution est lancée.

- En l'absence de <**nom de fichier**>, le premier programme rencontré est chargé,
  Dans ce cas, le message :

Searching apparaît sur l'écran indiquant la recherche du premier programme. Celui-ci trouvé, le message suivant apparaît :

Found : <**nom de fichier**> ← *1er enregistrement rencontré chargement en cours.*

OK ← *fin du chargement.*

- Avec le <**nom de fichier**> si le programme correspond au premier fichier rencontré le processus de recherche et de chargement est identique à celui expliqué ci-dessus.
  Si le programme ne correspond pas au premier fichier stocké les messages successifs suivants apparaissent sur l'écran.

Searching

Skip : <**nom de fichier**> ← *passage du 1er fichier*

Searching

Skip : <**nom de fichier**> ← *passage du 2e fichier*

Searching

.....

Found : <**nom de fichier**> ← *le fichier déterminé est trouvé ; chargement en cours.*

OK ← *arrêt du défilement de la bande ; fin de chargement.*

Si cette commande est incorporée à un programme, son exécution provoque le chargement d'un nouveau programme en remplacement du programme en mémoire. Dans ce cas, **Searching, Skip** et **Found** ne sont pas affichés. L'exécution du nouveau programme est lancée si l'option **R** est présente.

Exemples :

```
LOAD "PROG.2"
```

L'exécution ultérieure s'effectue en frappant la commande **RUN.**

```
LOAD "PROG.2",R
```

La fin de chargement le défilement de la bande magnétique cesse et l'exécution du programme s'accomplit.

```
LOAD "",R
```

Le premier programme rencontré est chargé et exécuté.

Remarque :

Si aucun programme n'est rencontré, la bande défile jusqu'à la fin et il faudra réinitialiser l'ordinateur.

MERGE

Syntaxe :

MERGE ["<descripteur de fichier>"[,R]]

<**descripteur de fichier**> est une constante chaîne entre guillemets, une variable chaîne ou une expression chaîne.

But :

Charge le programme désigné par <**nom de fichier**> (dans <**descripteur de fichier**>) et le fusionne à celui résidant en mémoire principale.

Le <**nom de fichier**> est le nom sous lequel le programme a été sauvegardé par l'instruction **SAVE** avec option A (programme stocké sous forme ASCII ou par l'instruction **LIST** "nom de fichier". Si le programme n'est pas sous forme ASCII, l'erreur **FM** (accès incorrect) est détectée.

Si le nom du programme est omis, le premier programme rencontré sur la cassette est chargé et fusionné s'il est sous forme ASCII.

Les lignes du programme lu sont interclassées avec celles du programme résidant en mémoire, d'après leurs numéros. Si les numéros sont identiques, la ligne du programme du fichier remplace celle du programme en mémoire.

Après exécution de cette commande, il y a retour au mode commande sauf si l'option **R** est demandée, auquel cas l'exécution est lancée depuis le début.

Exemple :

Supposons que le programme résidant en mémoire soit :

```
10 PRINT "Prix du livre: 20F"
20 INPUT "Quantité commandée";A
```

et que le programme ci-dessous, de nom "**PRIX. TTC**" soit enregistré sous forme ASCII :

```
100 X=A*20
110 PRINT "Prix de vente:"X;"F"
```

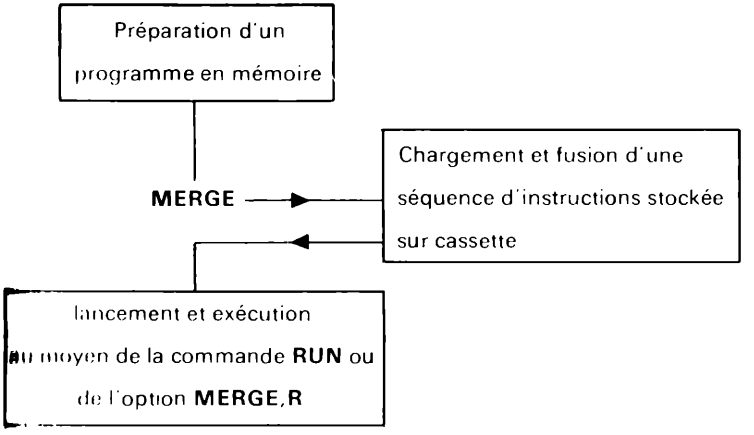
La frappe au clavier de **MERGE "PRIX. TTC"**, **R** effectue la fusion des deux programmes ci-dessus et le lancement de l'exécution du programme résultant :

```
Prix du livre: 20F
Quantité commandée? 100
Prix de vente: 2000 F
```

Remarque :

La commande **MERGE** permet également de constituer une bibliothèque de "sous-programmes" pouvant selon les besoins être chargés sélectivement en mémoire.

Le schéma général de travail est :



RUN

Syntaxe :

RUN "<descripteur de fichier>" [,R]

But :

Charge en mémoire et lance l'exécution du programme stocké sur le périphérique précisé dans le <**descripteur de fichier**> ou celui pris par défaut (lecteur-enregistreur de programmes). Dans ce dernier cas le <**descripteur de fichier**> se réduit à <**nom de fichier**>. Cette commande efface toutes les variables de l'ancien programme avant de lancer l'exécution et ferme tous les fichiers encore ouverts sauf si l'option **R** est indiquée : les fichiers restent alors ouverts.

Placée dans un programme, la commande **RUN** permet de mettre en œuvre une technique de "chaînage" de programmes rangés séquentiellement.

Exemples :

```
RUN "THOM1.JEU"

RUN ""
```

Le deuxième exemple charge le 1er programme rencontré sur la cassette et l'exécute.

Remarque :

Voir également chapitre II.

Syntaxe :

SAVE "<descripteur de fichier>" [, A / P]

descripteur de fichier> est une constante chaîne entre guillemets, une variable chaîne ou une expression chaîne.

But :

Sauvegarde du programme présent en mémoire.

descripteur de fichier> est de syntaxe identique à celle vue au § IV.2. Il se réduit à <**nom de fichier**> dans le cas d'utilisation du lecteur-enregistreur de programmes.

Si l'option **A** est précisée le programme est écrit et sauvegardé sous forme ASCII. Cette forme est nécessaire si l'on veut utiliser la commande **MERGE**.

Si l'option **P** est précisée, **LE PROGRAMME EST SAUVEGARDÉ EN MODE "PROTEGÉ", C'EST-A-DIRE ECRIT SOUS FORME CODÉE.**

Cette forme "**PROTEGÉE**" du programme sauvegardé en permet l'exécution mais **INTERDIT LE LISTAGE ET LES MODIFICATIONS AINSI QUE L'EXECUTION DE L'INSTRUCTION POKE ET DE LA FONCTION PEEK.**

Exemple :

```
SAVE "ANNEE",A
```

Conseil :

Etre très prudent avant l'exécution d'une instruction **SAVE** avec l'option **P** car toute modification et listage ultérieurs seront impossibles.

SKIPF

Syntaxe :

SKIPF["<nom de fichier>"]

But :

Cette instruction permet de positionner la bande de la cassette.
SKIPF sans nom de fichier provoque le saut du premier fichier rencontré. Si aucun fichier n'est rencontré, la bande défile jusqu'à la fin et il faudra réinitialiser l'ordinateur.

SKIPF <nom de fichier> provoque le défilement de la bande jusqu'à ce que le fichier indiqué soit "rencontré" et "sauté".

Dans le cas où le fichier indiqué n'existe pas, la bande défile jusqu'à la fin et il faudra réinitialiser l'ordinateur.

Le suffixe pris par défaut est **. BAS.**

Exemples :

```
SKIPF

SKIPF "DUMP.PRO"
```

Remarque :

L'ordinateur ne peut savoir, a priori, combien de fichiers sont contenus sur la cassette. Pour positionner la bande après le dernier fichier écrit, deux solutions sont possibles :

- le nom du dernier fichier est connu : dans ce cas, il suffira d'écrire :
  SKIPF <nom du dernier fichier>
- le nom du dernier fichier et le nombre de fichiers contenus sur la cassette ne sont pas connus : dans ce cas, il faudra exécuter la commande **SKIPF** suivie d'un nom de fichier inexistant afin de relever le nom des fichiers et en particulier le nom du dernier fichier. Ensuite il faudra :
- rembobiner la cassette
- réinitialiser l'ordinateur.
- exécuter la commande **SKIPF <nom du dernier fichier>**

Conseil :

Afin d'éviter de détruire un fichier par inadvertance, il est commode de noter sur la cassette les noms des fichiers dans l'ordre de stockage.

IV 4 FICHIERS DE DONNEES

Ce paragraphe concerne les fichiers de données à accès séquentiel

Un fichier séquentiel est constitué par une suite d'articles enregistrés par exemple sur cassette magnétique dans laquelle l'accès à l'un des articles s'effectue après lecture des précédents.

Notion de canal

A l'inverse des fichiers-programmes les fichiers de données ne se constituent pas en une seule opération de transfert mais généralement s'accroissent au fur et à mesure du développement du programme.

Il est donc indispensable de maintenir une communication permanente avec le fichier de données en lecture et en écriture entre le moment où le fichier est ouvert et celui où on le ferme parce qu'on n'en a plus besoin.

Cette communication durable s'appelle un canal logique.

La sauvegarde ou la relecture d'un fichier-programme ouvre et ferme son propre canal de façon transparente ce qui dispense l'utilisateur d'avoir à le faire, par contre toute opération d'entrée-sortie de donnée nécessite au préalable l'''ouverture'' du canal au moyen de l'instruction **OPEN**

La fin d'utilisation le canal doit être fermé au moyen de l'instruction **CLOSE**.

Le concept de canal logique associe un nombre compris entre 1 et 16 inclus à un nom de fichier ou de périphérique.

Un numéro de canal incorrect par exemple 17 est détecté par l'erreur **NU**.

Le canal ϕ est en permanence ouvert et affecté en entrée au clavier et en sortie à l'écran.

L'ensemble des instructions, commandes ou fonctions permettant l'ouverture, la fermeture des canaux et la création des fichiers de données est décrit ci-après.

Remarque :

L'exécution d'une instruction d'écriture sur fichier ne provoque pas systématiquement l'écriture sur cassette : le **TO7** attend que le tampon de sortie soit plein pour provoquer l'écriture. Par contre, l'exécution de l'instruction **CLOSE** provoque l'écriture du tampon de sortie même s'il n'a pas été rempli complètement.

De même en lecture, un tampon d'entrée est chargé au fur et à mesure des besoins.

La cassette ne défile donc pas systématiquement à chaque exécution d'une instruction d'entrée ou de sortie.

CLOSE

Syntaxe :

CLOSE [[#]<numéro de canal> [,[#]<numéro de canal>]...]

But :

Cette instruction ''ferme'' les canaux dont le numéro est indiqué et qui sont encore ouverts.

Dans cette instruction le numéro de canal est le numéro qui lui a été affecté lors de l'ouverture du canal (Cf. instruction **OPEN**).

L'instruction **CLOSE** sans numéro de canal ferme tous les canaux qui étaient ouverts juste avant son exécution. Ces canaux pourront à nouveau être ouverts au moyen d'une instruction **OPEN**, soit avec le même numéro soit avec un numéro différent.
De même, tout numéro de canal ainsi libéré peut servir à ouvrir à nouveau soit le même fichier, soit un fichier différent.

La fermeture d'un fichier séquentiel ouvert en écriture, provoque avant la fermeture proprement dite, l'écriture du ''tampon de sortie'' même si celui-ci n'est pas rempli.

L'instruction **END** et la commande **NEW** provoquent la fermeture de tous les fichiers encore ouverts avant leur exécution. Par contre, l'instruction **STOP** ne provoque pas la fermeture des fichiers.

Exemple :

```
1Ø OPEN"O",1,"ESSAI"        'ouverture du canal
                             de numéro 1 en mode O
        .
        .
5Ø PRINT#1,A$               'écriture de données
        .
        .
1ØØ CLOSE 1                 'fermeture du canal
        .
        .
```

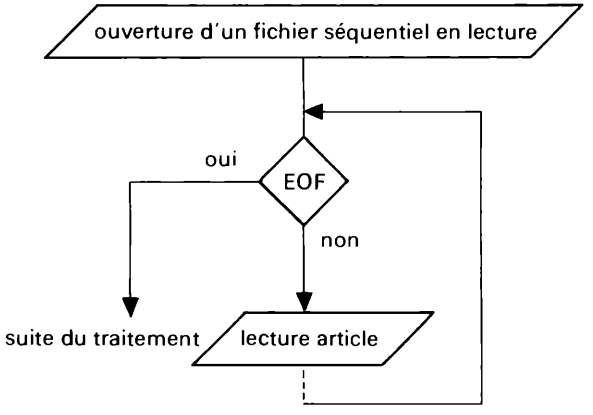
Syntaxe :

EOF (<numéro de canal>)

But :

Cette fonction donne la valeur **– 1** (correspondant à "**vrai**") si la fin d'un fichier séquentiel est atteinte sur le canal indiqué. Elle donne Ø si la fin de fichier n'est pas atteinte.

Les tests de fin de fichier, avant une instruction de lecture, évitent de tenter de lire un article inexistant, ce qui provoquerait l'erreur **IE**.



Exemple :

```
1Ø OPEN"I",1,"MESURES"   'ouverture du canal de
                          numéro 1 en mode I
2Ø N=Ø : M=Ø
3Ø IF EOF(1) THEN 8Ø     'test de fin de fichier
4Ø INPUT#1,XMES          'lecture des données et
                          affectation aux
                          variables
5Ø M=M+XMES
6Ø N=N+1
7Ø GOTO 3Ø
8Ø M=M/N
   PRINT"MOYENNE= ";M;" NOMBRE= ";N
```

INPUT

Syntaxe :

INPUT # <**numéro de canal**>, <**variable**> [,<**variable**>]...

But :

Lire les données d'un fichier séquentiel et affecter leur valeur aux <**variables**>. Les affectations aux variables s'effectuent successivement dans l'ordre d'apparition des données : la première donnée est affectée à la première variable, la deuxième à la seconde variable, etc. jusqu'à épuisement, le nombre de données enregistrées à lire étant égal au nombre de <**variable**>

- Si les variables sont plus nombreuses que les données, c'est-à-dire que le fichier ne contient plus de données pendant la lecture, l'erreur **IE** est détectée.

- Si les données sont plus nombreuses que les variables, elles sont affectées dans leur ordre d'enregistrement aux seules variables écrites dans **INPUT #**. Les autres pourront être utilisées lors d'une prochaine instruction **INPUT #**.

Les données de type chaîne doivent correspondre à des variables de même type dans la liste de variables figurant dans **INPUT.**

Par contre, une donnée de type numérique peut correspondre indifféremment à une variable numérique ou à une variable chaîne

Remarque :

Pour éviter toute tentative de lecture après une fin de fichier il est possible d'utiliser la fonction **EOF**.

Exemple :

```
10 OPEN"I",#1,"ESSAI.FIC"  'ouverture du canal
                            de numéro 1 en
                            mode I
20 PRINT"ETAT";TAB(15);"SUPERFICIE"
30 IF EOF(1) THEN END      'test de fin de
                            fichier
40 INPUT#1,NOM$,SURF       'lecture des données
                            et affectation
                            aux variables
50 PRINT NOM$;TAB(17);SURF
60 GOTO 30
```

INPUT$

Syntaxe :

INPUT$ (<I>[,[#]<numéro de canal J>])

But :

Cette fonction permet de lire à partir du clavier ou du canal de numéro **J** une chaîne de **I** caractères. Si le clavier est utilisé pour l'entrée, il n'y a pas d'écho sur l'écran c'est-à-dire que les caractères lus ne sont pas affichés. Tous les caractères de contrôle sont transmis y compris les caractères CNT C et STOP qui dans ce cas n'interrompent pas l'exécution du programme.

Exemple 1 :

```
1ØØ PRINT "DONNER VOTRE MOT DE PASSE"
11Ø A$=INPUT$(7)
12Ø IF A$<>"BONJOUR" THEN 6ØØ
                    'BONJOUR est le mot de passe
        .
        .
        .
        .
59Ø END
6ØØ PRINT "VOTRE MOT DE PASSE EST FAUX"
61Ø PRINT "RECOMMENCEZ"
62Ø GOTO 11Ø
```

Exemple 2 :

```
1ØØ PRINT"frapper S pour arrêter"
11Ø PRINT"frapper C pour continuer"
12Ø R$=INPUT$(1)
13Ø PRINT R$
14Ø IF R$="S" THEN END
15Ø IF R$<>"C" THEN 1ØØ
16Ø ...
```

Exemple 3 :

```
1ØØ OPEN"I",1,"FICHIER"   'ouverture du canal 1
                           en mode I
11Ø IF EOF(1) THEN 1ØØØ   'test de fin de fichier
12Ø A$=INPUT$(1,#1)
        .
        .
        .
```

Remarques :

Ainsi que l'on peut le voir, cette fonction permet de redonner le contrôle à l'ordinateur dès que le nombre de caractères demandé est lu sans attendre le caractère retour chariot.

Si on désire que la réponse de l'utilisateur soit visualisée, il faut ajouter une instruction **PRINT** (Cf. exemple 2).

LINE INPUT

Syntaxe :

LINEINPUT # <**numéro de canal**>, <**variable chaîne**>
ou
LINE INPUT # <**numéro de canal**>, <**variable chaîne**>

But :

Cette instruction permet de lire les données d'un fichier séquentiel comportant une séquence d'au plus 255 caractères composée de lettres, chiffres, signes de ponctuation, espace, etc. et de les affecter à <**variable chaîne**>.

La fin de la séquence affectée, est marquée par le caractère "**retour chariot**". Tous les caractères à l'exclusion du "**retour chariot**" sont affectés à la variable.

Le numéro du canal est celui sous lequel il a été ouvert par l'instruction **OPEN**.

Exemple :

```
1Ø OPEN"I",#1,"REGIONS"   'ouverture du canal 1
                           en mode I
2Ø LINE INPUT#1,A$        'lit les données et
                           les affecte à A$
3Ø PRINT A$               'affichage des données
```

Remarque :

Dans un fichier séquentiel, enregistré sur le lecteur-enregistreur de programmes par l'instruction **PRINT #**, les données sont toutes séparées par un caractère "**retour chariot**"

Dans ce cas **LINE INPUT #** relira les données une par une.

O = Ordinateur → périphérique
I = Périphérique → ordinateur

OPEN

Syntaxe :

OPEN { "O" / "I" }[#]<**numéro de canal**>,"<**descripteur de fichier**>"

avec **O** signifiant **output** c'est-à-dire sortie : le sens des informations est ORDINATEUR vers PERIPHERIQUE

I signifiant **input** c'est-à-dire entrée : le sens des informations est PERIPHERIQUE vers ORDINATEUR.

- <**numéro de canal**> est compris entre **1** et **16** inclus.
- <**descripteur de fichier**> est une constante chaîne entre guillemets, une variable chaîne ou une expression chaîne.

But :

Cette instruction permet d'ouvrir le ou les canaux à utiliser pour effectuer les opérations d'entrées-sorties.

Le canal Ø est en permanence ouvert et affecté en entrée au clavier et en sortie à l'écran. Un numéro de canal incorrect est détecté par l'erreur **NU**.

Exemples :

Clavier et écran

Pour utiliser le clavier ou l'écran avec un numéro de canal égal à 5 par exemple, le canal est ouvert au moyen de l'une des instructions :

- **clavier**

```
OPEN "I",5,"KYBD:"
```

- **écran**

```
OPEN "O",5,"SCRN:"
```

- **utilisation d'une imprimante à interface parallèle.**

```
OPEN "O",3,"LPRT:(8Ø)"
```

Le canal 3 est associé à l'imprimante à interface parallèle dont les lignes contiennent jusqu'à 8Ø caractères

Utilisation de la voie série.

```
OPEN "O",#1,"COMM:(48132) RESULTAT.MAT"
```

permet l'ouverture en sortie (sens ordinateur vers périphérique) de la voie série avec les caractéristiques suivantes :

- numéro de canal : 1
- vitesse : 12ØØ bauds
- nombre de bits transmis pour un caractère : 8 bits
- 132 caractères par ligne
- nom de fichier : RESULTAT.MAT

Utilisation du lecteur-enregistreur de programmes

Le <**descripteur de fichier**> se réduit à <**nom de fichier**> car le lecteur-enregistreur est le périphérique pris par défaut.

```
1Ø OPEN "I",#8,"ESSAI"
```

Ouvre le canal de numéro 8 en entrée, pour la lecture de données en provenance du lecteur-enregisteur de programmes. **ESSAI** est le nom de fichier.

PRINT

PRINT # USING

Syntaxe :

PRINT # <N° de canal>[,[USING<m>;] [<n> [,<p>]...]]

où **m** est une expression chaîne
n et **p** sont des expressions

But :

Cette instruction permet l'écriture de données pour la création de fichiers séquentiels.

N° de canal > est le numéro sous lequel le canal a été ouvert par une instruction **OPEN** en mode "**O**".

Si le terme **[USING** <**m**> **;]** est présent, l'écriture des données s'effectue selon le format décrit dans <**m**>.
La valeur de <**m**> contient des descripteurs qui indiquent le format désiré, de manière similaire à l'instruction **PRINT USING** (Cf. chapitre II)

Les termes <**n**> et <**p**> peuvent être numériques ou de forme chaîne de caractères : ils définissent ce qu'il faut écrire sur le fichier.

L'instruction **PRINT #** écrit les données dans le fichier dans la même forme qu'elles le seraient sur l'écran par l'instruction **PRINT**.
Toutefois, l'écriture d'une ligne complète ne provoque pas de saut de ligne dans le cas d'utilisation d'une imprimante.

Exemple :

```
100 OPEN"O",#1,"LPRT:"
                        'ouverture du canal
                         de numéro 1
110 FOR I=1 TO 5
120 READ A$,N           'lecture des données DATA
130 PRINT#1,A$,N        'écriture des données
140 NEXT I
150 CLOSE#1             'fermeture du canal numéro 1
160 DATA ALABAMA,133667
170 DATA ARIZONA,295014
180 DATA CALIFORNIE,41015,COLORADO,2700
190 DATA CONNECTICUT,12973
```

```
RUN
ALABAMA          133667
 RIZONA          295014
 LIFORNIE        41015
   RADO          2700
     TICUT       12973
```

Résultat obtenu sur une imprimante connectée en parallèle

Remarque :

Sur le lecteur-enregistreur, **PRINT** # enregistre un caractère "**retour chariot**" entre chaque variable

IV.5 FICHIERS BINAIRES

Il est possible de stocker et de charger des fichiers "**binaires**" au moyen des instructions **SAVEM** et **LOADM**.

Un fichier binaire peut contenir un programme "**binaire**" proprement dit ou encore une image écran

Syntaxe :

SAVEM"<descripteur de fichier>", <adr.1>, <adr.2>, <adr.d'exéc.>

But :

Cette instruction permet de sauvegarder un programme binaire.

adr. 1> représente l'adresse de début du programme.
adr. 2> représente l'adresse de la fin du programme.
adr. d'exéc.> est l'adresse de lancement du programme.

Cette instruction permet en outre de sauvegarder une image écran sur cassette.
On peut :

sauvegarder le dessin proprement dit
sauvegarder les couleurs.

La mémoire d'écran se trouve aux adresses hexadécimales **&H4000 à &H5F3F**. Pour sauvegarder le dessin (ou formes) on pourra utiliser la séquence suivante :
POKE &HE7C3, 1 OR PEEK (&HE7C3) Sélection de la mémoire de formes.
SAVEM "<nom de fichier>", &H4000, &H5F3F, 0

Si on désire sauvegarder les couleurs de l'écran il faudra créer un fichier supplémentaire au moyen de la séquence suivante :
POKE &HE7C3, 254 AND PEEK (&HE7C3) sélection de la mémoire couleur.
SAVEM "<nom de fichier>", &H4000, &H5F3F, 0

Exemple :

```
10 CLS                                    'efface l'écran
20 BOXF (100, 10) - (10, 100), 1          'rectangle rouge
30 BOX (150, 60) - (110, 200), 2          'rectangle vert
40 POKE &HE7C3, 1 OR PEEK (&HE7C3)
50 SAVEM"FORME", &H4000, &H5F3F, 0
                                          'sauvegarde"FORME"
60 POKE &HE7C3, 254 AND PEEK (&HE7C3)
70 SAVEM"COULEUR", &H4000, &H5F3F, 0
                                          'sauvegarde"COULEUR"
```

LOADM

Syntaxe :

LOADM ["<descripteur de fichier>"][,[<décalage>][,R]]

But :

Permet de charger un programme binaire ou une image écran stocké sur cassette.
<**décalage**> est une expression numérique dont la valeur indique de combien le chargement doit être translaté en mémoire

Pour le chargement de l'image "forme" d'un écran, il faut exécuter la séquence suivante en mode programme.
POKE &HE7C3, 1 OR PEEK (&HE7C3) *sélection mémoire "forme"*
LOADM "<nom de fichier>"

Le chargement des couleurs se fera au moyen de la séquence suivante.
POKE &HE7C3, 254 AND PEEK (&HE7C3) *sélection mémoire "couleur"*
LOADM "<nom de fichier>"

Si l'option **R** est présente, l'exécution binaire est lancée à l'adresse d'exécution définie lors du **SAVEM** Le décalage décale aussi l'adresse d'exécution.

V - GRAPHISME

Le présent chapitre traite des possibilités graphiques du **TO 7.** Un paragraphe est consacré au crayon optique et à sa mise en œuvre.

V 1 LE SYSTEME D'ECRAN

L'écran est divisé en 2ØØ lignes, une ligne étant composée de 4Ø segments de huit points. A l'intérieur de chaque segment les points peuvent appartenir au graphisme ou au fond. Un segment ne peut se voir attribuer que deux couleurs :

une couleur pour les points "graphisme"

une couleur pour les points "fond".

Le tableau ci-dessous donne les codes correspondants pour chaque couleur

Couleur	Code couleur du graphisme	Code couleur du fond
NOIR	Ø	– 1
ROUGE	1	– 2
VERT	2	– 3
JAUNE	3	– 4
BLEU	4	– 5
MAGENTA (Violet)	5	– 6
CYAN (Bleu clair)	6	– 7
BLANC	7	– 8

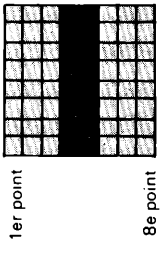
Dans un segment les couleurs adoptées sont celles qui ont été déterminées lors de la programmation des derniers points "graphisme" et "fond" de ce segment.

Si la programmation d'un point d'un segment ne comporte pas d'attribut couleur, il est traité comme un point "graphisme" et la couleur prise par défaut à cet instant pour ce point et pour tous les points "graphisme" de ce segment est celle qui a été choisie précédemment pour les caractères (instructions **SCREEN, COLOR** ou action sur la touche **INITIALISATION**).

D'un segment de huit points à un autre le choix de la paire de couleurs est entièrement libre. Il est possible, sans modifier les couleurs, d'obtenir un autre dessin en changeant la répartition des points "graphisme" et "fond"

Exemple :

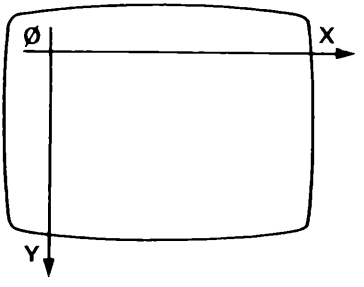
```
5 CLS                    'effacement de l'écran
1Ø SCREEN Ø,7,7          'caractères noirs sur
                          fond blanc
2Ø FOR Y=4Ø TO 47         '8 segments superposés
30 PSET(24Ø,Y),1         'premier point "graphisme"
                          rouge
4Ø PSET(241,Y),1         'deuxième point "graphisme"
                          rouge
5Ø PSET(242,Y),1         'troisième point "graphisme"
                          rouge
6Ø PSET(243,Y),-6        'quatrième point "fond"
                          magenta
7Ø PSET(244,Y),-4        'cinquième point "fond"
                          jaune:détermine la couleur
                          "fond" du segment
8Ø PSET(245,Y),4         'sixième point "graphisme"
                          bleu
9Ø PSET(246,Y),4         'septième point "graphisme"
                          bleu
1ØØ PSET(247,Y)          'huitième point "graphisme"
                          noir par défaut:détermine
                          la couleur"graphisme"du
                          segment
11Ø NEXT
12Ø END

RUN
```



***N.B.** : A la ligne 8Ø la programmation d'un point "graphisme" en bleu change provisoirement en bleu les trois premiers points qui étaient rouges*
A la ligne 1ØØ l'absence du paramètre couleur crée un point "graphisme" de couleur noire (couleur par défaut indiquée dans **SCREEN***) et colorie de nouveau les points 1, 2, 3, 6 et 7 en noir*

V.2 COORDONNEES D'UN POINT DE L'ECRAN

Les coordonnées utilisées dans les instructions graphiques sont mesurées à partir des axes de référence représentés dans la figure ci-dessous. Ces axes non matérialisés coïncident avec les bords gauche et supérieur de la surface utilisable de l'écran.



Le **TO 7** fonctionne en permanence en mode graphique c'est-à-dire que l'affichage des caractères obéit à toutes les règles du graphisme : les caractères sont dessinés en "graphisme" dans des matrices 8 × 8 (8 segments superposés). L'entourage du caractère appartient au fond

Les coordonnées d'un point de l'écran peuvent être spécifiées en mode "**graphique**" ou en mode "**caractère**" :

en mode "**graphique**" la résolution est de 64ØØØ points (2ØØ lignes de 32Ø points)

en mode "**caractère**" l'écran est divisé en 25 rangées de 4Ø caractères (une rangée comporte 8 lignes).

Les valeurs admises pour X et Y sont donc

$$\left.\begin{array}{l} \emptyset \leq X \leq 319 \\ \emptyset \leq Y \leq 199 \end{array}\right\} \text{ mode "**graphique**"}$$

$$\left.\begin{array}{l} \emptyset \leq X \leq 39 \\ \emptyset \leq Y \leq 24 \end{array}\right\} \text{ mode "**caractère**"}$$

X et **Y** sont des expressions numériques de type entier, réel ou double précision. Si nécessaire la valeur des coordonnées est convertie en entier par arrondi

Dans le cas où les valeurs attribuées à **X** et **Y** seraient supérieures aux limites indiquées ci-dessus, le **TO 7** prendrait en compte :

$$\left.\begin{array}{l} X = 319 \\ Y = 199 \end{array}\right\} \text{ en mode "**graphique**"} \qquad \left.\begin{array}{l} X = 39 \\ X = 24 \end{array}\right\} \text{ en mode "**caractère**"}$$

L'erreur **FC** est détectée si **X** et **Y** prennent des valeurs négatives.

Il est à noter que les coordonnées spécifiées en mode "**graphique**" et celles spécifiées en mode "**caractère**" sont dans un rapport de 8

V.3 INSTRUCTIONS COMMANDES ET FONCTIONS GRAPHIQUES

Les instructions, commandes et fonctions graphiques sont décrites dans les pages suivantes

Syntaxe :

$$\left\{\begin{matrix}\textbf{BOX}\\ \textbf{BOXF}\\ \textbf{LINE}\end{matrix}\right\}\textbf{[(X1, Y1)] – (X2, Y2)}\left\{\begin{matrix}\text{"}<\textbf{car}>\text{"}\ [,<\textbf{attribut 1}>]\\ [,<\textbf{attribut 2}>]\end{matrix}\right\}$$

$$\textbf{PSET (X, Y)}\left\{\begin{matrix}\text{"}<\textbf{car}>\text{"}\ [,<\textbf{attribut 1}>]\\ [,<\textbf{attribut 2}>]\end{matrix}\right\}$$

Xn, **Yn** = coordonnées du point ou du caractère.

But :

$\left\{\begin{matrix}\textbf{BOX}\\ \textbf{BOXF}\end{matrix}\right\}$ trace un rectangle dont les côtés sont parallèles à ceux de l'écran. Les coordonnées des sommets opposés **(X1, Y1)** et **(X2, Y2)** déterminent les dimensions du rectangle. Avec l'instruction **BOX** seuls les côtés du rectangle sont de la couleur déterminée par l'attribut tandis que **BOXF** trace un rectangle dont toute la surface est coloriée.

LINE permet de tracer un segment de droite entre les deux points de coordonnées **(X1, Y1)** et **(X2, Y2)**.

PSET positionne sur l'écran un point ou un caractère dont la couleur est définie par l'attribut.

Le type des expressions numériques **X, Y, X1, Y1, X2, Y2** est indiqué au paragraphe V.2.

Si le couple **(X1, Y1)** est omis, ces coordonnées sont remplacées soit :

- par les valeurs **(X2, Y2)** de la dernière instruction **BOX**, **BOXF** ou **LINE** exécutée.
- par les valeurs **(X, Y)** de la dernière instruction **PSET**.
- par les valeurs **(Ø, Ø,)** si aucune instruction graphique n'a encore été exécutée.

L'argument <**car**> représente une expression chaîne dont le premier caractère sera utilisé pour l'affichage. Ce peut être une chaîne entre " ou une variable chaîne.

Sa présence indique que les coordonnées doivent être spécifiées selon le mode "**caractère**" et son omission qu'elles doivent être données selon le mode "**graphique**".

En mode "**caractère**", < **attribut 1** > est composé de trois paramètres. Ceux-ci sont définis dans l'instruction **COLOR**.

En mode "**graphique**", < **attribut 2** > est une expression numérique qui peut prendre une valeur de – 8 à 7 :

- une valeur négative détermine l'emploi d'une couleur ''de fond''.
- une valeur de Ø à 7 correspond à l'emploi d'une couleur ''graphisme''.

L'absence d'attribut couleur entraîne l'utilisation de la couleur par défaut.

Remarques :

1 - Les instructions **BOX, BOXF, LINE** et **PSET** intéressent la totalité de l'écran même si une fenêtre de travail plus petite que celui-ci a été définie. Elles ne sont pas affectées par l'utilisation de l'instruction **ATTRB**.

2 - En mode graphique, l'attribut couleur n'agit que sur l'instruction en cours d'exécution et le curseur n'est pas déplacé.

Exemple :

```
1Ø SCREEN,7,7  'fond blanc
2Ø CLS         'effacement de l'écran
3Ø BOX(Ø,Ø)-(319,199),Ø
               'délimitation de la fenê-
                tre de travail par un
                trait noir
4Ø LINE-(Ø,Ø),-4
               'tracé d'un segment dans
                la couleur de fond jaune
                (origine 319,199 par
                 défaut)
5Ø LINE(Ø,199)-(319,Ø),Ø
               'tracé d'un segment de
                couleur"graphisme"noir
6Ø BOX(1Ø4,8)-(216,48),3
               'tracé d'un rectangle dont
                les côtés sont jaunes
7Ø BOXF(1Ø4,144)-(216,192),3
               'tracé d'un rectangle
                colorié en jaune
8Ø BOX(1,9)-(12,15)"*",3,Ø
               'tracé d'un rectangle par
                des caractères"*"jaunes
                sur fond noir
9Ø BOXF(27,9)-(38,15)"$",Ø,3
               'tracé d'un rectangle
                rempli de caractères"$"
                noirs sur fond jaune
```

COLOR

Syntaxe :

COLOR [X] [, [Y] [, Z]]

X = couleur caractère ; **Y** = couleur fond ; **Z** = inversion

But :

Changer les règles d'utilisation des couleurs de l'écran pour les prochains caractères qui seront affichés.

X indique la couleur des caractères qui seront édités.

Y précise la couleur de fond.

Z peut prendre les valeurs **Ø** ou **1**. Si **Z** est omis, les paramètres **X** et **Y** ne sont pas modifiés. Si **Z = Ø** ou **1**, les couleurs sont permutées.

Les valeurs admises pour **X** et **Y** sont :

Ø	noir	**4**	bleu
1	rouge	**5**	magenta (violet)
2	vert	**6**	cyan (bleu clair)
3	jaune	**7**	blanc

Exemples :

```
COLOR 2,Ø   'caractères verts sur fond noir
COLOR Ø,7   'caractères noirs sur fond blanc
COLOR ,,1   'caractères blancs sur fond noir
```

Remarque :

Pour le **TO7-70**, voir l'annexe G.

DEFGR$

Syntaxe :

DEFGR$(I) = < liste d'entiers >

où **I** est une expression dont la valeur peut varier de **Ø à 127**

< **liste d'entiers** > est composé de 8 expressions dont les valeurs après arrondi peuvent varier de **Ø à 255,**

But :

Cette instruction permet de définir des graphismes ''utilisateur'' qui pourront ensuite être utilisés comme les caractères du jeu standard de l'ordinateur

I désigne le numéro de ce graphisme.

La définition de ces graphismes se fait dans une matrice 8 × 8

Pour définir un graphisme, on peut opérer de la façon suivante .

- dessiner le graphisme dans une matrice comportant 8 lignes, chaque ligne étant à son tour composée de 8 petits rectangles.
- marquer chaque petit rectangle faisant partie du graphisme
- pour chaque ligne, calculer la valeur correspondant au nombre binaire obtenu en prenant :
  - 1 pour chaque rectangle marqué
  - Ø pour les rectangles non marqués
- définir **GR$(I)** en plaçant dans la liste la séquence des huit nombres ainsi obtenus, séparés par des virgules, dans l'ordre, du haut vers le bas du graphisme. La valeur des 8 nombres peut être exprimée en binaire, en octal, en décimal ou en hexadécimal.

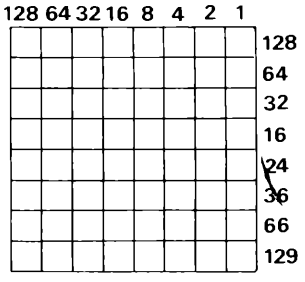
Il est possible de définir ainsi jusqu'à 128 caractères graphiques ''utilisateur'' dont les numéros iront de Ø à 127

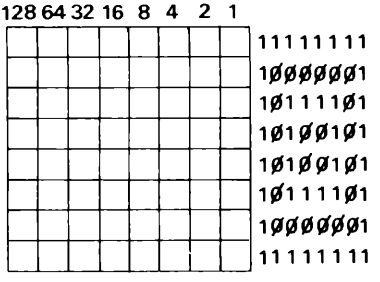
Ces graphismes pourront être désignés par **GR$(I)** ou par **CHR$(128 + I)**.

Ils pourront être utilisés en sortie, par des instructions **PRINT** ou **PRINT USING** avec sortie sur écran.

Ces graphismes ne peuvent pas être lus en entrée puisque le clavier ne permet pas l'entrée de caractères non standards

Exemple :






```
1Ø ATTRB 1,1      'caractères double taille
2Ø CLEAR,,2       'réserve de la place en mémoire
3Ø DEFGR$(Ø)=128,64,32,16,24,36,66,129
                  'caractère"lambda"(numération
                   décimale)
4Ø DEFGR$(1)=&B11111111,&B1ØØØØØØØ1,&B1Ø1111Ø1,
&B1Ø1ØØ1Ø1,&B1Ø1ØØ1Ø1,&B1Ø1111Ø1,&B1ØØØØØØØ1,
&B11111111        'caractère"carré"(numération
                   binaire)
5Ø PRINTGR$(Ø);   'lambda
6Ø PRINTGR$(1)    'carré
7Ø END
```

Remarque :

Cette instruction ne peut fonctionner que si de la place a été réservée en mémoire par l'intermédiaire de l'instruction **CLEAR**.

POINT

Syntaxe :

POINT(X, Y)

X = abscisse du point ; **Y** = coordonnée du point.

But :

Cette fonction donne une valeur numérique comprise entre − 8 et 7 qui indique la couleur du point de l'écran de coordonnées **(X, Y)**.

X et **Y** représentent des expressions numériques dont les valeurs, après arrondi éventuel à l'entier le plus proche, doivent se trouver dans l'intervalle **(Ø, 319)** pour **X** et **(Ø, 199)** pour **Y**.

Cette fonction renvoie une valeur positive ou nulle si le point appartient au graphisme :

Ø	pour noir	**4**	pour bleu
1	pour rouge	**5**	pour magenta (violet)
2	pour vert	**6**	pour cyan (bleu clair)
3	pour jaune	**7**	pour blanc

Elle renvoie une valeur négative si le point appartient au fond :

− 1	noir	**− 5**	bleu
− 2	rouge	**− 6**	magenta
− 3	vert	**− 7**	cyan
− 4	jaune	**− 8**	blanc

Exemple :

```
1ØØ X=POINT(2ØØ,1ØØ)
```

Remarque :

Pour le TO7-70, voir l'annexe G.

SCREEN

Syntaxe :

SCREEN [A] [,[B] [,[C] [, J]]]

A = couleur caractère ; **B** = couleur fond ; **C** = couleur tour ; **J** = inverser

Une restriction à cette syntaxe : elle ne doit pas se terminer par une virgule.

But :

Cette instruction permet de changer instantanément les couleurs de la fenêtre de travail et de l'entourage de l'écran dans les conditions suivantes :

- le paramètre **A** indique la couleur des caractères (couleur,graphisme)
- le paramètre **B** indique la couleur de fond
- le paramètre **C** indique la couleur du cadre (appelé aussi entourage)
- la présence du paramètre **J** indique qu'il faut inverser les couleurs du graphisme et du fond.

Les expressions **A**, **B** et **C** doivent prendre des valeurs comprises entre Ø et 7 comme pour l'instruction **COLOR**.

L'expression **J** doit prendre pour valeur Ø ou 1, après arrondi éventuel à l'entier le plus proche.

Exemple :

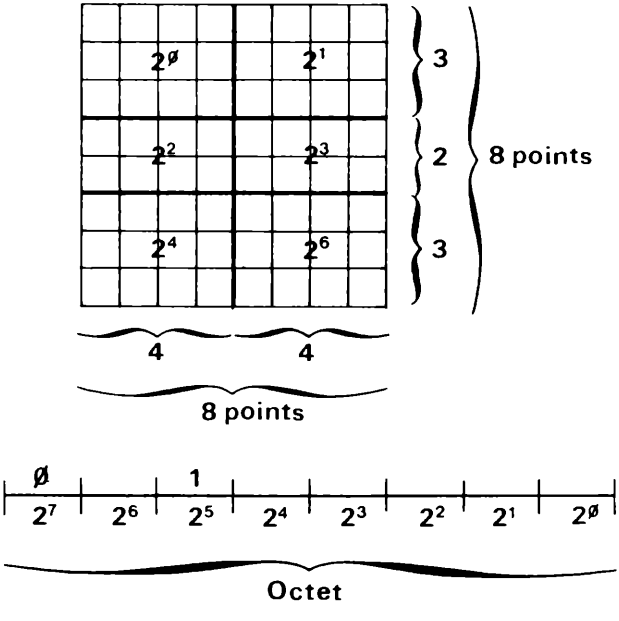
```
SCREEN 1,Ø,5      'caractères rouges sur fond
                   noir avec cadre violet
SCREEN 2,6,5,1    'caractères cyans sur fond
                   vert avec cadre violet
SCREEN ,,1        'caractères cyans sur fond
                   vert avec cadre rouge
```

Remarques :

Ne pas confondre l'instruction **SCREEN** avec la fonction du même nom, pour le **TO7-70** voir l'annexe G.

V.4 LES CARACTERES SEMI-GRAPHIQUES TELETEL

Des caractères aux normes TELETEL sont disponibles. Ils s'inscrivent dans des carrés de 8 × 8 points. Ces caractères sont divisés en 6 zones, chacune d'entre elles étant représentée par un chiffre binaire occupant un rang déterminé dans un octet.



Si le chiffre binaire est Ø, la zone correspondante est de "couleur fond".

Si le chiffre binaire est 1, la zone est de "couleur caractère".

La valeur de l'octet détermine la forme du caractère à afficher. Il est ainsi possible de représenter 64 caractères différents (voir tableau ci-après). Les chiffres de rang 2^5 et 2^7 gardent respectivement les valeurs 1 et Ø.

Ces caractères ne sont pas modifiés par l'instruction **ATTRB** L'affichage s'effectue grâce à des instructions **PRINT**.

1° - passage au mode semi-graphique par **CHR$ (14)**.

2° - détermination du caractère par **CHR$(X)** où **X** représente la valeur de l'octet correspondant au caractère Cette valeur peut être exprimée en hexadécimal, en octal, en binaire ou en décimal

3° - Retour au mode normal par **CHR$ (15)**.

Exemple :

```
PRINT CHR$(14)     'passage en mode
                   semi-graphique
PRINT CHR$(102)    'forme du caractere
PRINT CHR$(15)     'retour au mode normal
```

Il est possible de n'utiliser qu'une seule instruction **PRINT**.

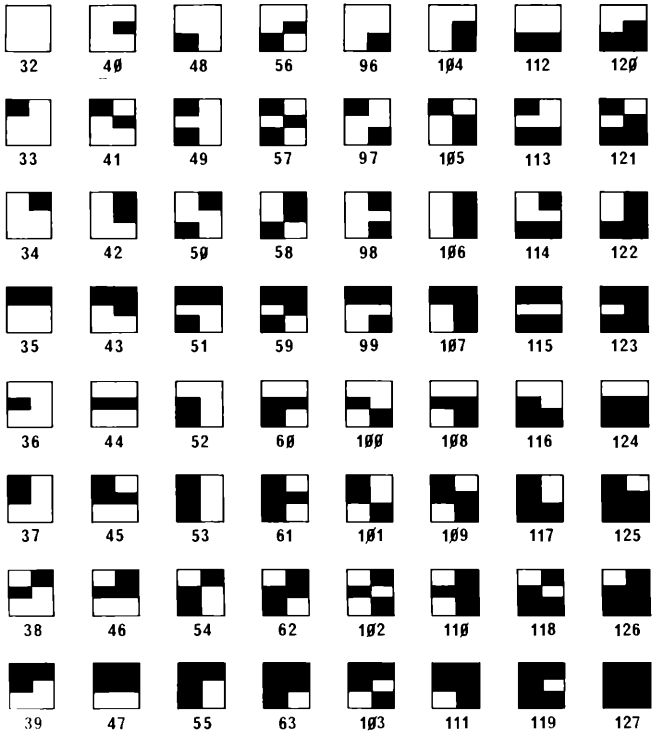
Exemple :

```
PRINT CHR$(14);CHR$(102);CHR$(15)
```

Remarque :

Chacun des codes peut être remplacé par la lettre correspondante : **CHR$ (102)** → **"f"**. Le retour au mode commande restaure le mode normal

CARACTERES SEMI-GRAPHIQUES TELETEL



V.5 AFFICHAGE D'UN TEXTE DANS UN RECTANGLE

Il est nécessaire dans un premier temps de s'assurer que le rectangle tracé par des instructions **BOX** ou **BOXF** est suffisamment grand pour contenir le texte à afficher.

Les dimensions du texte sont exprimées en mode "**caractère**", celles du rectangle peuvent être données en mode "**graphique**" ou en mode "**caractère**". Les coordonnées d'un point en mode "**graphique**" correspondent aux coordonnées de ce point en mode "**caractère**", multipliées en huit.

Le premier caractère du texte à afficher sera positionné dans le rectangle par une instruction **LOCATE**.

Exemple :

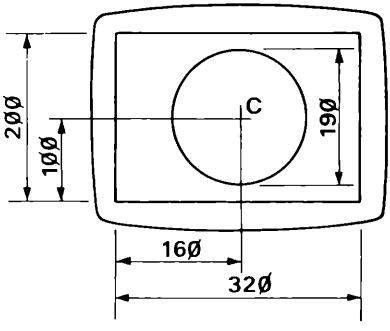
```
5 SCREEN Ø,7,7      'caractères noirs sur
                     fond blanc
1Ø CLS              'effacement de l'écran
2Ø BOXF(64,56)-(232,96),3
                    'tracé d'un rectangle jaune
3Ø L=(232-64)/8     'longueur du rectangle
4Ø H=(96-56)/8      'hauteur du rectangle
5Ø CHAINE$="AFFICHAGE RESERVE"
                    'texte à afficher
6Ø LONG=LEN(CHAINE$)
                    'longueur du texte a afficher
7Ø N=1              'nombre de lignes du texte
8Ø X=64/8+(L-LONG)/2
                    'pos. horizontale du curseur
9Ø Y=56/8+(H-N)/2   'pos. verticale du curseur
1ØØ LOCATE X,Y      'positionnement du curseur
11Ø PRINT CHAINE$   'affichage du texte
12Ø LOCATE Ø,15     'curseur hors du rectangle
13Ø END
```

6. TRACE DE COURBES

Il est nécessaire dans un premier temps de calculer les coordonnées de différents points et d'effectuer ensuite une transformation afin de rendre ces coordonnées compatibles avec les dimensions de l'écran.

Les courbes seront tracées au moyen des instructions **PSET** (point par point) ou **LINE** (par segments de droite).

Le tracé d'un cercle (centré par rapport à l'écran) présente un exemple de la méthode de travail



Les coordonnées d'un cercle sont :

X = R*COS (TETA)
Y = R*SIN (TETA)

Le diamètre du cercle est limité par la plus petite dimension de l'écran soit 2ØØ points dans le sens vertical Choisissons donc un cercle de rayon

$$\frac{19Ø}{2} = 95 \text{ points}$$

Le centre du cercle aura pour coordonnées $X_c = 16Ø$ et $Y_c = 1ØØ$. Par conséquent

X = R*COS (TETA) + 16Ø
Y = R*SIN (TETA) + 1ØØ

TETA variant de Ø à 2*PI

6.1 Tracé point par point

L'instruction utilisée est **PSET**.

Exemple :

```
1ØØ CLS
11Ø R=95
12Ø FOR TETA=Ø TO 2*3.14159 STEP Ø.Ø4
13Ø X=R*COS(TETA)+16Ø
14Ø Y=R*SIN(TETA)+1ØØ
15Ø PSET(X,Y),1    'tracé par points de
                    couleur rouge
16Ø NEXT TETA
```

V.6.2 Tracé par segments de droite

- Positionner un point du cercle, par exemple celui correspondant à **TETA = Ø** dont les coordonnées sont (R + 16Ø, 1ØØ).

- Utiliser l'instruction **LINE** sans préciser les coordonnées d'origine du segment. Ainsi le cercle est tracé à l'aide de petits segments de droite.

Exemple :

```
1ØØ CLS
11Ø R=95
12Ø PSET(R+16Ø,1ØØ),1
                   'initialisation du tracé
13Ø FOR TETA=Ø TO 2*3.14159 STEP Ø.Ø1
14Ø X=R*COS(TETA)+16Ø
15Ø Y=R*SIN(TETA)+1ØØ
16Ø LINE-(X,Y),1   'tracé par segments de
                    couleur rouge
17Ø NEXT TETA
```

V.7 COMMENT DESSINER AVEC LE CRAYON OPTIQUE

Les exemples suivants montrent comment associer les instructions relatives au crayon optique.

```
10 INPUT I                          ' choix de la couleur du tracé
20 CLS                              ' effacement de l'écran
30 SCREEN, 7                        ' fond blanc
40 INPUTPEN X,Y                     ' lecture des coordonnées
50 PSET (X,Y), I                    ' affichage d'un point de couleur I
                                    ' initialisant le tracé
60 INPEN X1, Y1                     ' lecture des coordonnées
70 IF NOT PTRIG THEN 40
80 IF X1 = 0 OR Y1 = 0 THEN 60
                                    ' élimine les lectures
                                    ' erronées
90 LINE -(X1,Y1), I                 ' tracé de couleur I
100 GOTO 60
```

Afin d'obtenir un tracé plus régulier, il est préférable de calculer les coordonnées d'un point moyen situé entre le dernier point affiché et le point visé par le crayon. Ces coordonnées seront utilisées dans une instruction **PSET** ou **LINE**.

```
10 CLS
20 SCREEN 7
30 INPUT I
40 INPUTPEN X1%, Y1%
50 PSET (X1%, Y1%), I
60 INPEN X1, Y1
65 IF NOT PTRIG THEN 40
70 IF X1 = 0 OR Y1 = 0 THEN 60
80 X1% = (X1% + X1)/2               ' calcul du point moyen
                                    ' horizontal
90 Y1% = (Y1% + Y1)/2               ' calcul du point moyen
                                    ' vertical
100 LINE -(X1%, Y1%), I
110 GOTO 60
```

V.8 COMMENT "GOMMER" AVEC LE CRAYON OPTIQUE

Un trait étant composé de points, "**gommer**" consiste à placer ces points dans la couleur de fond.

Pour cela on utilise les instructions **INPEN** ou **INPUTPEN** et **PSET** avec un attribut couleur correspondant à la couleur de fond. L'exemple suivant suppose que la couleur de fond est cyan.

```
5 SCREEN,6          'fond de couleur cyan
1Ø INPUTPEN X,Y     'lecture des coordonnees
2Ø IF X<Ø OR Y<Ø THEN 1Ø
                    'élimine les lectures erronées
3Ø PSET(X,Y),-7     'affichage d'un point couleur
                     fond
4Ø GOTO 1Ø
```

Afin d'obtenir une gomme qui permette d'effacer plus rapidement, il est possible d'utiliser l'instruction **PSET** en mode caractère.

```
1Ø LOCATE Ø,Ø,Ø          'suppression du curseur
2Ø INPUTPEN X,Y          'lecture des coordonnées
3Ø IF X<Ø OR Y<Ø THEN 2Ø
                         'élimine les lectures
                          erronées
4Ø PSET(X/8,Y/8)" "      'affichage du caractère
                          "espace"
5Ø GOTO 2Ø
```

VI - SON

VI.1 INTRODUCTION

Un générateur de sons permettant la composition musicale, sur une plage de cinq octaves, est incorporé à l'appareil. Les sons transmis par le haut-parleur du téléviseur peuvent comporter des notes :

- avec ou sans dièse ou bémol
- avec durée réglable (de la ronde à la triple croche)
- avec tempo réglable
- avec attaque réglable

Il est également possible de lire la piste son du lecteur-enregistreur de programmes. Le son est restitué par le haut-parleur du téléviseur.

Dans chaque cas, l'utilisateur peut régler l'intensité sonore en agissant sur le bouton de réglage du téléviseur.

VI.2 INSTRUCTION PLAY

Cette instruction permet de "jouer" des airs de musique connus et de composer sa propre musique.

Syntaxe :

PLAY <expression chaîne> [, <expression chaîne>] ...

Les chaînes de caractères incluses dans <expression chaîne> doivent avoir un contenu bien déterminé sinon l'erreur **FC** est détectée :

· **chaîne de caractères>** = **<élément>** **[[;]<élément>]** ...

élément> = { <note> [{ # / b }] / <silence> / <attribut> }

· **attribut>** = A <ent> / T <ent> / O <ent> / L <ent>

note> = DO / RE / MI / FA / SO / LA / SI

silence> = P

ent> = constante entière dont la valeur doit se situer dans une plage qui est

Signification	attribut	Plage de l'attribut	valeur implicite
Attaque	A	Ø – 255	Ø
Longueur	L	1 – 96	24 (noire)
Octave	O	1 – 5	4
Tempo	T	1 – 255	5

A = Ø correspond à un son continu. Une valeur élevée pour l'attaque rend le son rapidement amorti.

L indique la durée relative des notes ou des silences qui suivent. La durée minimum pour un son est obtenue avec la valeur 1 pour le paramètre de L et la durée maximum est obtenue avec la valeur 96.

L = 96 correspond à une ''ronde''
L = 48 correspond à une ''blanche''
L = 24 correspond à une ''noire''
L = 12 correspond à une ''croche''
L = 6 correspond à une ''double croche''
L = 3 correspond à une ''triple croche''

L = 72 correspond à une ''blanche pointée''
L = 36 correspond à une ''noire pointée''
L = 18 correspond à une ''croche pointée''
L = 9 correspond à une ''double croche pointée''

Toutes les durées intermédiaires sont également possibles. Par exemple, il est possible d'obtenir des notes ''triolets'' :

L = 64 correspond à une ronde triolet
L = 32 correspond à une blanche triolet
L = 16 correspond à une noire triolet
L = 8 correspond à une croche triolet
L = 4 correspond à une double croche triolet
L = 2 correspond à une triple croche triolet

O indique le numéro de l'octave à utiliser. La valeur 1 correspond à l'octave la plus grave et 5 à l'octave la plus aiguë.

P la présence de P donne un silence de longueur **L**.

T règle la vitesse absolue de l'exécution, c'est-à-dire qu'il change toute la durée des notes dans une même proportion.
La valeur T = 1 correspond au tempo le plus rapide et T = 255 correspond au tempo le plus lent.

La présence des attributs A, L, O et T agit sur les notes qui suivent jusqu'à ce qu'une autre valeur viennent modifier la valeur attribuée.

Une note suivie de la lettre minuscule b correspond à un bémol. Si une note est suivi du symbole #, la note correspond à un dièse.

Remarquons que le bémol ne doit pas changer de lui même l'octave. Ainsi, dans l'exemple suivant :

```
PLAY"O3DODOb"
```

Le second DO est plus aigu que le premier. Pour obtenir un DOb plus grave que le DO précédent, il faudrait écrire :

```
PLAY"O3DOO2DOb03"
```

Exemple :

```
 Ø A$= "O5DOREMIDOREPREMIFAFAMIPDOREMIDO
REPREMIFASODO"
2Ø PLAY A$
3Ø PLAY A$+A$
```

Exemple de composition aléatoire

```
Ø CLEAR 5ØØ
1Ø DIM A$(6)
2Ø FOR I=Ø TO 6
3Ø  READ A$(I)
4Ø NEXT I
5Ø DATA DO,RE,MI,FA,SO,LA,SI
1ØØ B$=""
11Ø FOR I=1 TO 1Ø
12Ø X=INT(6*RND)
13Ø B$=B$+A$(X)+"O"+RIGHT$(STR$(INT(5*RND+1)),1)
14Ø NEXT I
15Ø PLAY B$
16Ø GOTO 1ØØ
```

VII – EXEMPLES DE PROGRAMMES

VII.1 INTRODUCTION

Ce chapitre présente quelques programmes mettant en œuvre les instructions et fonctions BASIC présentées dans les chapitres précédents. Il s'agit de programmes simples, construits dans un but pédagogique et destinés à aider le lecteur à concevoir ensuite des programmes plus élaborés.

VII.2 PROGRAMME ELEMENTAIRE DE LISTAGE PHYSIQUE DES MEMOIRES

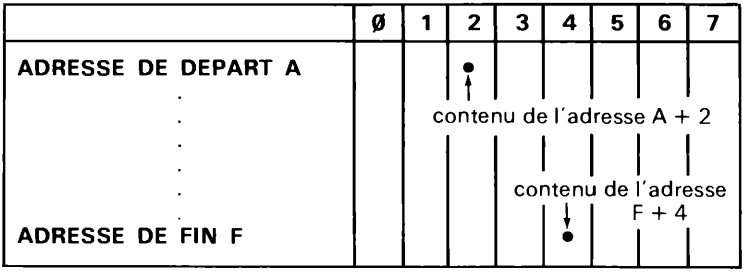
Le programme suivant permet d'effectuer le "**dump**" de portions de mémoire centrale en utilisant la fonction **PEEK**.

```
10 'PROGRAMME ELEMENTAIRE DE LISTAGE MEMOIRE
20 '
90 A$="#####      ####"   'image de
                           l'instruction 130
100 INPUT"ADRESSE DEBUT ET FIN";A1,A2
125 PRINT:PRINT"ADRESSE CONTENU":PRINT
110 FOR I=A1 TO A2
120 X=PEEK(I)              'contenu des adresses
                           mémoires
130 PRINT USING A$;I,X
140 NEXT

RUN
ADRESSE DEBUT ET FIN? 0,10

ADRESSE CONTENU

    0        32
    1        66
    2        65
    3        83
    4        73
    5        67
    6        32
             77
             73
             67
```

VII.3 PROGRAMME DE LISTAGE MEMOIRE EN BASE 16

Pour effectuer un listage du contenu d'une zone de la mémoire centrale en base 16, on peut utiliser la séquence ci-dessous donnée à titre d'exemple.

On affiche 8 valeurs par ligne ; pour cela il suffit de construire un tableau ayant l'allure suivante :



Pour obtenir un tel tableau on utilise l'instruction **PRINT USING** qui permet des éditions soignées. Cela conduit à utiliser les images contenues dans les variables **A$**, **B$** et **C$**.

```
1ØØ CLS:SCREEN 2,Ø,5
11Ø INPUT"ADRESSE DE DEPART";A
12Ø INPUT"ADRESSE DE FIN   ";F
13Ø A$="*****"      } programme simple ne comportant aucun test
14Ø B$=" %%"        } sur A et F - Images utilisées pour l'affichage.
15Ø C$="       Ø  1  2  3  4  5  6  7"
16Ø PRINT C$:PRINT
17Ø FOR X=A TO F STEP 8
18Ø   PRINT USING A$;X,
19Ø   FOR I=Ø TO 7
2ØØ     Y=PEEK(X+I)        <- lecture de l'octet
21Ø     T$=HEX$(Y)         <- conversion en hexadécimal
22Ø     IF LEN(T$)=1 THEN T$="Ø"+T$   <- ajout d'un zéro si longueur 1 pour cadrage à droite
23Ø     PRINT USING B$;T$,
24Ø   NEXT I
25Ø   PRINT
26Ø NEXT X

RUN

ADRESSE DE DEPART? 2ØØØ
ADRESSE DE FIN   ? 2Ø48
      Ø  1  2  3  4  5  6  7

 2ØØØ CØ 1Ø 27 1Ø 57 81 BD 1Ø
 2ØØ8 27 5A 61 4C 27 2F 8D Ø6
 2Ø16 BD Ø8 1A C6 29 8C C6 28
 2Ø24 8C C6 2C E1 9F 61 B9 26
 2Ø32 Ø2 ØE B2 C6 Ø2 7E Ø3 53
 2Ø4Ø 86 7D BD Ø8 2Ø 7E 25 F3
 2Ø48 BD ØA 48 9F 57 8D CD 26
OK
```

VII.4 COMPTAGE DES VOYELLES DANS UNE PHRASE

Le programme de ce paragraphe permet de compter le nombre d'occurrences de chaque voyelle dans une phrase introduite au clavier.

Le tableau V$ contient les voyelles :

V$(1) contient la voyelle A
V$(2) contient la voyelle E
V$(3) contient la voyelle I
V$(4) contient la voyelle O
V$(5) contient la voyelle U
V$(6) contient la voyelle Y

La variable **P$** contiendra la phrase introduite au moyen de l'instruction de la ligne **18Ø INPUT P$**. Si la phrase doit contenir des virgules, cette instruction devra être remplacée par :

LINE INPUT P$

La recherche des voyelles se fait au moyen de deux boucles emboîtées :

- **C$** contiendra successivement chacun des caractères de la phrase.
- les lignes 23Ø à 26Ø permettent de savoir si **C$** contient une voyelle.
- le tableau V contient le nombre d'apparition de chaque voyelle.

```
10Ø 'COMPTAGE DES VOYELLES DANS UNE PHRASE
11Ø 'INITIALISATION DES VARIABLES
12Ø FOR I=1 TO  6
13Ø   READ V$(I)
14Ø   V(I)=Ø
15Ø NEXT
16Ø DATA A,E,I,O,U,Y
165 CLS
17Ø PRINT"Ecrivez une phrase en majuscule"
18Ø INPUT P$
19Ø L=LEN(P$)
2ØØ 'ANALYSE DE LA PHRASE
21Ø FOR I= 1 TO L
22Ø   C$=MID$(P$,I,1)          <- Extraction d'un caractère
23Ø   FOR J=1 TO 6
24Ø     IF C$=V$(J) THEN V(J)=V(J)+1   <- Test : C$ est-il
25Ø   NEXT J                              une voyelle ?
26Ø NEXT I                          Si oui incrémentation V(J)
 7Ø 'EDITION DU RESULTAT
 8Ø PRINT: PRINT "Cettre phrase contient :"
 9Ø PRINT: PRINT:ATTRB 1,1
 ØØ FOR I= 1 TO 6
 1Ø   IF V(I)>Ø THEN PRINT V(I); V$(I)   <- Affichage
 2Ø                                         des résultats
 3Ø       Ø,Ø
```

VII.5 EDITION DE L'ALPHABET ASCII

Ce programme permet d'éditer l'alphabet ASCII à partir du code 32 qui correspond au caractère "espace". L'édition est faite sur 3 colonnes.

```
10 CLS
20 F$="##   %%     ##  %%     ###  %%"
30 FOR I=1 TO 32
40   I1=31+I
50   I2=63+I
60   I3=95+I
70   A$=CHR$(I1) : B$=CHR$(I2)
75   C$=CHR$(I3)
80   PRINT USING F$;I1,A$,I2,B$,I3,C$
90 NEXT I
```

division du tableau en 3 zones

La fonction CHR$ donne le caractère correspondant au code donné comme paramètre

```
RUN
32        64  @     96  `
33  !     65  A     97  a
34  "     66  B     98  b
35  #     67  C     99  c
36  $     68  D    100  d
37  %     69  E    101  e
38  &     70  F    102  f
39  '     71  G    103  g
40  (     72  H    104  h
41  )     73  I    105  i
42  *     74  J    106  j
43  +     75  K    107  k
44  ,     76  L    108  l
45  -     77  M    109  m
46  .     78  N    110  n
47  /     79  O    111  o
48  0     80  P    112  p
49  1     81  Q    113  q
50  2     82  R    114  r
51  3     83  S    115  s
52  4     84  T    116  t
53  5     85  U    117  u
54  6     86  V    118  v
55  7     87  W    119  w
56  8     88  X    120  x
57  9     89  Y    121  y
58  :     90  Z    122  z
59  ;     91  [    123  {
60  <     92  \    124  |
61  =     93  ]    125  }
62  >     94  ↑    126  ~
63  ?     95  _    127  ■

OK
```

VII.6 JEU C'EST PLUS, C'EST MOINS

Le programme suivant permet de jouer avec l'ordinateur. L'ordinateur commence par demander un nombre entier entre 5 et 2Ø.

Puis il demande un nombre A que l'on pourra choisir de l'ordre de 1ØØ ou de 1ØØØ. Il faudra ensuite trouver par essais successifs le nombre choisi par l'ordinateur entre Ø et A. Le programme affiche à chaque essai la fourchette dans laquelle se trouve le nombre, il compte en outre le nombre d'essais effectués.

```
5Ø SCREEN 2,Ø,5
1ØØ PRINT"DONNER UN NOMBRE ENTIER "
11Ø INPUT"ENTRE 5 ET 2Ø";B
12Ø INPUT"DONNER LE NOMBRE MAXIMUM";A
13Ø FOR I=1 TO B
14Ø X=RND                    ← détermination d'un
15Ø NEXT I                     nombre aléatoire
16Ø N=INT(A*RND+.5):C=Ø:D=A
17Ø CLS:I=Ø
18Ø LOCATE Ø,1
19Ø PRINT TAB(7);"TROUVER UN NOMBRE ENTRE"
2ØØ ATTRB 1,1:LOCATE Ø,5:PRINT C;TAB(12);"ET"
21Ø LOCATE 3Ø,5:PRINT D
22Ø LOCATE Ø,12
23Ø INPUT"VOTRE CHOIX";X
24Ø I=I+1
25Ø ON SGN(N-X)+2 GOTO 3ØØ,39Ø,35Ø
3ØØ IF X<D THEN D=X
31Ø LOCATE 3Ø,5:ATTRB 1,1
32Ø PRINT D
33Ø GOTO 22Ø
35Ø IF X>Ø THEN C=X
36Ø LOCATE Ø,5:ATTRB 1,1
37Ø PRINT C
38Ø GOTO 22Ø
39Ø LOCATE Ø,19:COLOR 4,Ø:ATTRB 1,1
4ØØ PRINT"TROUVE APRES";I;TAB(12);"ESSAIS"
41Ø ATTRB Ø,Ø
```

ANNEXE A

REPRESENTATION DES VARIABLES EN MEMOIRE

1 - VARIABLES ENTIERES

Ces variables sont représentées en "binaire pur", en mode complément à 2 sur 2 octets, le premier octet contenant les bits de poids forts et le second les bits de poids faibles.

Ainsi leur plage de variation est donc limitée de − 32 768 à + 32 767.

Pour lire le contenu d'une variable entière X%, on pourra donc écrire :

```
A = VARPTR (X%)
X1 = PEEK (A)
X2 = PEEK (A + 1)
PRINT X1, X2, 256*X1 + X2
```

2 - VARIABLE REELLE (encore appelée "simple précision")

Ces variables sont représentées sous forme de 4 octets en mode virgule flottante :

Le 1er octet contient l'exposant,
Le 2e octet contient les bits de poids forts de la mantisse
Le 3e octet contient les bits de poids moyens de la mantisse
Le 4e octet contient les bits de poids faibles de la mantisse

3 - VARIABLE DOUBLE PRECISION

La mantisse se distingue de celle des variables réelles, par le fait qu'elle contient 7 octets au lieu de 3.

4 VARIABLE CHAINE

Le premier octet indique la longueur de la chaîne.

Le second octet contient les bits de poids forts de l'adresse du premier octet de la chaîne.

Le troisième contient les bits de poids faibles de l'adresse du premier octet de la chaîne.

Ainsi pour lire individuellement les différents caractères d'une chaîne, on pourra écrire :

```
10 A$="ABCDEF"
20 A=VARPTR(A$)
30 L=PEEK(A)
40 IF L=0 THEN STOP
50 A1=256*PEEK(A+1)+PEEK(A+2)
60 FOR I=0 TO L-1
70 PRINT PEEK(A1+I)
80 NEXT I
```

ANNEXE B

FONCTIONS MATHEMATIQUES

Pour simuler les fonctions mathématiques qui ne sont pas disponibles, l'utilisateur peut écrire un petit sous-programme composé généralement d'une instruction d'affectation et de l'instruction RETURN, pour obtenir la valeur désirée.

Le tableau suivant donne une liste des formules qui peuvent être utilisées.

FONCTION	FORMULE DE CALCUL
sécante	1/COS(X)
cosécante	1/SIN(X)
cotangente	1/TAN(X)
sinus hyperbolique	(EXP(X) − EXP(−X))/2
cosinus hyperbolique	(EXP(X) + EXP(−X))/2
tangente hyperbolique	1 − 2*EXP(−X)/(EXP(X) + EXP(−X))
arcsinus hyperbolique	LOG(X + SQR(X*X + 1))
arccosinus hyperbolique	LOG(X + SQR(X*X − 1))
arctangente hyperbolique	LOG((1 + X)/(1 − X))/2

arctangente
- pour $-1 <= X <= 1$ utiliser la formule

(((((A9*X2 + A7)*X2 + A5)*X2 + A3)*X2 + A1)*X avec

X2 = X*X
A1 = .999866
A3 = −.33Ø2995
A5 = .18Ø141
A7 = .Ø85133
A9 = .Ø2Ø8351

- pour $X < -1$:
  - calculer l'arctangente Y1 de 1/X
  - l'arctangente Y de X s'obtient ensuite par
    Y = −1.57Ø796
- pour $X > 1$:
  - calculer l'arctangente Y1 de 1/X
  - l'arctangente Y de X s'obtient ensuite par
    Y = 1 57Ø796 − Y1

arcsinus : Pour $Ø <= X <= 1$ on peut obtenir l'arcsinus au moyen de la formule suivante :

PI/2 − SQR(1 − X)*(((A3*X + A2)*X + A1)*X + AØ)

avec : AØ = 1.57Ø7288
A1 = −.2121144
A2 = .Ø74261Ø
A3 = −.Ø187293

Exemple : Utilisation de la fonction arctangente pour $-1 \leqslant X \leqslant 1$

```
10 A$="**.**   **.*******"
20 A1=.999866
30 A3=-.330295
40 A5=.180141
50 A7=-.085133
60 A9=.020835
70 FOR X=-1 TO 1 STEP .1
80 X2=X*X
90 GOSUB 200
100 PRINT USING A$;X,ATN
110 NEXT X
120 END
200 ATN=((((A9*X2+A7)*X2+A5)*X2+A3)*X2+A1)*X
210 RETURN

RUN
-1.00   -0.7854100
-0.90   -0.7328160
-0.80   -0.6747510
-0.70   -0.6107190
-0.60   -0.5404090
-0.50   -0.4636510
-0.40   -0.3805180
-0.30   -0.2914610
-0.20   -0.1973870
-0.10   -0.0996580
 0.00    0.0000001
 0.10    0.0996582
 0.20    0.1973870
 0.30    0.2914610
 0.40    0.3805180
 0.50    0.4636510
 0.60    0.5404100
 0.70    0.6107200
 0.80    0.6747510
 0.90    0.7328160
OK
```

ANNEXE C

ERREURS DETECTEES

Code	numéro	
AO	52	Fichier déjà ouvert (Already Open)
BD	58	Mauvaises données dans le fichier (Bad Data)
BS	9	Indice en dehors des limites (Subscript out of range)
CN	17	Suite d'exécution impossible (Can't continue) – arrêt sur erreur – modification du programme après une interruption par CNT C
DD	1Ø	Définition multiple ou définition d'un tableau après l'avoir utilisé avec un dimensionnement implicite (redimensioned array)
DS	56	Commande directe dans un fichier **ASCII** en cours de chargement (Direct Statement)
DU	6Ø	Périphérique non disponible (Device unavailable)
FC	5	Argument incorrect dans l'appel d'une fonction ou de certaines instructions (Illegal function call)
FD	55	Descripteur de fichier incorrect (Bad file descriptor)
FM	51	Mode d'accès au fichier incorrect (Bad file mode)
FN	23	Instruction **FOR** sans instruction **NEXT** (For without Next)
ID	12	Instruction interdite en mode immédiat (Illegal Direct)
IE	54	Tentative de lecture d'un fichier après la fin de fichier pour éviter cette erreur, utiliser la fonction **EOF** (Input past End)
IO	53	Erreur d'entrée/sortie (Input/Output)
IU	59	Périphérique en cours de fonctionnement (Device in use)
LS	15	Chaîne de caractères trop longue (String too long)
MO	22	Opérande manquant (Missing Operand)
NF	1	Instruction **NEXT** sans instruction **FOR** ou **NEXT** avec variable ne correspondant pas à l'instruction **FOR** exécutée (Next without For)
NO	57	Fichier non ouvert (Not Open)
NR	19	Instruction **RESUME** manquante dans le sous-programme de traitement des erreurs (No Resume)
NU	5Ø	Numéro de fichier incorrect (Not in use)
OM	7	Dépassement de capacité en mémoire centrale (Out of Memory)
OD	4	Instruction **READ** alors que les données des instructions **DATA** sont épuisées (Out of Data)
OS	14	Manque de place pour les chaînes (Out of string Space)
OV	6	Dépassement de capacité par valeur supérieure (OVerflow)
PP	61	Programme protégé (Protected Program)

RE	**2Ø**	Instruction **RESUME** alors qu'il n'y a pas eu d'erreur détectée (Resume without Error).
Redo		Un type de donnée ne correspond pas à celui indiqué dans **INPUT**
RG	**3**	Instruction **RETURN** alors que **GOSUB** n'a pas été exécuté (Return without Gosub)
SN	**2**	Erreur de syntaxe (Syntax error)
ST	**16**	Expression sur chaîne trop complexe (String formula Too complex)
TM	**13**	Non correspondance de type (Type Mismatch)
UE		Erreur non définie (Undefined Error)
UF	**18**	Fonction utilisateur non définie (Undefined user Fonction)
UL	**8**	Numéro de ligne inexistant (Undefined Line)
/Ø	**11**	Division par zéro
WE	**25**	**WEND** sans **WHILE (WEND** without **WHILE)**
WH	**24**	**WHILE** sans **WEND (WHILE** without **WEND)**

Numéro	Code	Numéro	Code
1	NF	19	NR
2	SN	2Ø	RE
3	RG	22	MO
4	OD	23	FN
5	FC	24	WH
6	OV	25	WE
7	OM	5Ø	NU
8	UL	51	FM
9	BS	52	AO
1Ø	DD	53	IO
11	/Ø	54	IE
12	ID	55	FD
13	TM	56	DS
14	OS	57	NO
15	LS	58	BD
16	ST	59	IU
17	CN	6Ø	DU
18	UF	61	PP

ANNEXE D

MOTS RESERVES

Un nom de variable ne doit ni être un mot réservé ni commencer par un mot réservé.

ABS
AND
ASC
ATTRB
AUTO

BEEP
BOX

CDBL
CHR$
CINT
CLEAR
CLOSE
CLS
COLOR
CONSOLE
CONT
COS
CSNG
CSRLIN

DATA
DEF
DELETE
DIM

ELSE
END
EOF
EQV
ERL
ERR
ERROR
EXEC
EXP

FIX
FN
FOR
FRE

GO
GR$

HEX$

IF
IMP
INKEY$
INPEN
INPUT
INSTR
INT

LEFT$
LEN
LET
LINE
LIST
LOAD
LOCATE
LOG

MERGE
MID$
MOD
MOTOR

NEW
NEXT
NOT

OCT$
OFF
ON
OPEN
OR

PEEK
PEN
PLAY
POINT
POKE
POS
PRINT
PSET
PTRIG

READ
REM
RESTORE
RESUME
RETURN
RIGHT$
RND
RUN

SAVE
SCREEN
SGN
SIN
SKIPF
SPC(
SQR
STEP
STICK
STOP
STRIG
STR$
SUB

TAB(
TAN
THEN
TO
TROFF
TRON

UNMASK
USING
USR

VAL
VARPTR

WAIT
WEND
WHILE

XOR

ANNEXE E

JEU DE CODES ASCII

Code décimal		Code décimal		Code décimal	
ØØ	NUL	42	*	85	U
Ø1		43	+	86	V
Ø2	STOP touche STOP clavier	44		87	W
Ø3	BREAK (CNT C)	45	–	88	X
Ø4		46	.	89	Y
Ø5		47	/	9Ø	Z
Ø6		48	Ø	91	[
Ø7	SONNETTE	49	1	92	
Ø8	BS (Recul arrière) ← clavier	5Ø	2	93	]
Ø9	HT (Tabulation H) → clavier	51	3	94	^
1Ø	LF (Saut de ligne) ↓ clavier	52	4	95	▯
11	VT (Tabulation V) ↑ clavier	53	5	96	▯
12	FF (Saut de page) RAZ clavier	54	6	97	a
13	CR (Retour à la ligne) ENTREE	55	7	98	b
14	SO Mode semi-graphique	56	8	99	c
15	SI Mode alphanumérique	57	9	1ØØ	d
16		58		1Ø1	e
17	DC1 Clignotement curseur	59		1Ø2	f
18	DC2 Répétition	6Ø	<	1Ø3	g
19		61	=	1Ø4	h
2Ø	DC4 Arrêt curseur	62	>	1Ø5	i
21		63	?	1Ø6	j
22	SS2 Touche caractères	64	@	1Ø7	k
	accentués	65	A	1Ø8	l
23		66	B	1Ø9	m
24	CAN Efface la fin de la ligne	67	C	11Ø	n
25		68	D	111	o
26		69	E	112	p
27	ESC Appel d'une séquence	7Ø	F	113	q
28	INS (INS clavier)	71	G	114	r
29	DEL (EFF clavier)	72	H	115	s
3Ø	RS Touche clavier ↵	73	I	116	t
31	US Séparateur d'article	74	J	117	u
32	Espace	75	K	118	v
33	!	76	L	119	w
34	"	77	M	12Ø	x
35	#	78	N	121	y
36	$	79	O	122	z
37	%	8Ø	P	123	▯
38	&	81	Q	124	▯
39	'	82	R	125	▯
4Ø	(	83	S	126	▯
41	)	84	T	127	■

Remarque : CNT force à Ø le bit 6 de la touche frappée en même temps

ANNEXE E bis

CODES ECRAN SPECIAUX

OBTENUS PAR : PRINT CHR$(22) ; CHR$(<code>)

Code décimal	Résultat
35	£
36	$
38	#
44	←
45	↑
46	→
47	↓
48	°
49	±
56	÷
6Ø	¼
61	½
62	¾
1Ø6	Œ
122	œ
65	`
66	´
67	^
72	¨
75	ç

Remarque : les caractères 65, 66, 67, 72 et 75 doivent être suivis d'une minuscule (c pour le code 75)

ANNEXE F

ORGANISATION DE LA MEMOIRE

Adresse (hexadécimale)	Contenu
ØØØØ – 3FFF	Cartouche "mémo 7" enfichable (16 k)
4ØØØ – 5FFF	Mémoire écran 8 k (double)
6ØØØ – 6ØFF	Réservé au système (page Ø)
61ØØ – DFFF	Libre pour mémoire utilisateur
EØØØ – FFFF	8 k système

MEMOIRE DU TO7-70

Dans sa version de base, le TO7-70 dispose de 48 K de mémoire vive utilisateur, dont seuls 32 K sont accessibles simultanément

La partie fixe de 16 K se trouve située de l'adresse 6ØØØ à l'adresse 9FFF, et la partie commutable se situe de AØØØ à DFFF Vous pouvez commuter celle-ci avec les 16 autres K disponibles · vous disposerez donc de deux blocs.

En cas d'ajout de l'extension-mémoire 64 K, vous disposerez de quatre blocs supplémentaires, aux mêmes adresses.

Ce BASIC ne sait pas gérer les blocs et n'utilise donc que 32 K de mémoire vive utilisateur sur les 48 disponibles Il ne peut pas gérer le ou les cinq blocs supplémentaires qui ne sont accessibles que par un sous-programme assembleur

Dans ce cas, pensez à faire un CLEAR, &H9FFF ou à recommuter le bloc Ø à la fin de votre traitement. .

Le sous-programme suivant vous permet de commuter le bloc de votre choix :

ROUTINE DE COMMUTATION DES BLOCS

Entrée · A = Numéro de bloc (Ø à 5)
Sortie · bloc commuté
Détruit · aucun registre
Longueur : 3Ø octets

```
        = ØØØØ'          COMMUT EQU   *
ØØØØ' 34 56                     PSHS  D,X,U
ØØØ2' CE E7CØ                   LDU   #$E7CØ
ØØØ5' E6 4B                     LDB   11,U
ØØØ7' C4 FB                     ANDB  #$FB
ØØØ9' E7 4B                     STB   11,U
ØØØB' 8E ØØ18'                  LDX   #TAB
ØØØE' A6 86                     LDA   A,X
ØØ1Ø' A7 49                     STA   9,U
ØØ12' CA Ø4                     ORB   #$Ø4
ØØ14' E7 4B                     STB   11,U
ØØ16' 35 D6                     PULS  D,X,U,PC
        = ØØ18'          TAB    EQU   *
ØØ18' ØF 17 E7 67               FCB   $ØF,$17,$E7,
      A7 27                           $67,$A7, $27
                                END
```

ANNEXE G

1) Le **TO7-70** dispose de 16 couleurs Fond et Forme pour les caractères.

Les instructions **SCREEN** et **COLOR** n'acceptent que des arguments inférieurs à 8.

Vous pouvez accéder aux 8 couleurs supplémentaires avec les ordres suivants :

soit **F** = 112 : couleur caractère ou forme
= 120 : couleur fond
= 128 : couleur tour

et **C** = Ø : gris
= 1 : rose
= 2 : vert clair
= 3 : jaune poussin
= 4 : bleu ciel
= 5 : rose parme
= 6 : cyan clair
= 7 : orange

L'instruction SCREEN pour le Fond et la Forme se fera par :

PRINT CHR$(27)CHR$(32)CHR$(F + C)

L'instruction COLOR et le tour de l'écran se fera par :

PRINT CHR$(27)CHR$(F + C)

La Fonction **POINT** retournera des valeurs comprises entre – 16 et 15 selon les couleurs :

Fond	Forme	
– 16	15	orange
– 15	14	cyan clair
– 14	13	rose parme
– 13	12	bleu ciel
– 12	11	jaune poussin
– 11	10	vert clair
– 10	9	rose
– 9	8	gris

2) Sur le TO7-70, l'instruction **SCREEN** supporte un cinquième argument, qui commande l'incrustation d'images vidéo. Cet argument peut prendre la valeur Ø ou 1, Ø commandant le mode normal et 1 commandant le mode incrusté. Dans le mode incrusté, tout point noir de l'écran est remplacé par l'image vidéo

Achevé d'IMPRIMER
Sur les PRESSES de l'IMPRIMERIE
I.G.R. au 3I/7/I984
Imprimé en FRANCE

BASIC D'UN SEUL COUP D'ŒIL

Nota : les instructions ou fonctions indiquées en *italiques* sont spécifiques du BASIC Disque et ne sont pas traitées dans ce manuel. Pour plus de détails, consultez "le BASIC D.O.S. du T07" aux Editions CEDIC/TO TEK International.

ABS 110
AND 21
ASC 111
ATTRB 26
AUTO 28

BACKUP
BEEP 30
BOX 187
BOXF 187

CDBL 112
CHR$ 113
CINT 115
CIRCLE
CLEAR 32
CLOSE 170
CLS 34
COLOR 189
CONSOLE 35
CONT 37
COPY
COS 116
CSNG 117
CSRLIN 118
CVD
CVI
CVS

DATA 38
DEFDBL 40
DEFGR$ 190
DEF FN
DEFINT 40
DEFSNG 40
DEFSTR 40
DEFUSR 42
DELETE 45
DEVICE
DIM 46
DIR
DRAW
DSKF
DSKI$
DSKINI
DSKO$

END 48
EOF 171
EQV 21
ERL 49
ERR 49
ERROR 51
EXEC 53
EXP 119

FIELD
FILES
FN 120
FOR...TO...STEP 54
FRE 121

GET
GET#
GOSUB...RETURN 57
GOTO 59
GR$ 190

HEX$ 122

IF...THEN...ELSE 61
IMP 21
INKEY$ 123
INPEN 63
INPUT 65
INPUT# 172
INPUT$ 174
INPUTPEN 63
INPUTWAIT 68
INSTR 124
INT 126

KILL

LEFT$ 127
LEN 128
LET 69
LINE 187
LINEINPUT 70
LINEINPUT# 176
LIST 159
LOAD 162
LOADM 182
LOC
LOCATE 72
LOF
LOG 129
LSET

MERGE 164
MID$ 73-130
MOD 19
MKD$
MOTOROFF 76
MOTORON 76
NAME
NEXT 54
NEW 77
NOT 21

OCT$ 131
ON ERROR GOTO 78
ON...GOSUB 80
ON...GOTO 80
ONPEN GOSUB 82
ONPEN GOTO 82
OPEN 177
OPEN "D"
OR 21
PAINT
PEEK 132
PEN 84
PLAY 201
POINT 192
POKE 86
POS 133
PRINT 87
PRINT# 179
PRINT#USING 179
PRINT USING 90
PSET 187
PTRIG 135
PUT
PUT#

READ 97
REM 99
RENUM
RESTORE 100
RESUME 101
RETURN 57
RIGHT$ 136
RND 137
RSET
RUN 102-166

SAVE 167
SAVEM 181
SCREEN 139-193
SCREEN PRINT 104
SGN 141
SIN 142
SKIPE 168
SPACE
SPC 143
SQR 144
STICK 145
STOP 105
STR$ 147
STRIG 146
STRING$
SWAP

TAB 148
TAN 150
TROFF 106
TRON 106

UNLOAD
UNMASK 107
USR 42

VAL 151
VARPTR 152
VERIFYON
VERIFYOFF

WAIT 108
WRITE

XOR 21